

UNIVERSIDAD POLITÉCNICA DE MADRID
E.T.S. DE INGENIERÍA DE SISTEMAS INFORMÁTICOS
TRABAJO FIN DE GRADO
GRADO EN TECNOLOGÍAS PARA LA SOCIEDAD DE LA INFORMACIÓN

Desarrollo de una aplicación en Python para el estudio y programación de SmartCards **SLE5528 y SLE5542**

Desarrollado por: David Balenzategui García

Dirigido por: José Gutiérrez Fernández

Madrid, 5 de noviembre de 2025

Desarrollo de una aplicación en Python para el estudio y programación de SmartCards SLE5528 y SLE5542

Desarrollado por: David Balenzategui García

Dirigido por: José Gutiérrez Fernández

Trabajo Fin de Grado, 5 de noviembre de 2025

E.T.S. de Ingeniería de Sistemas Informáticos

Campus Sur UPM, Carretera de Valencia (A-3), km. 7

28031, Madrid, España

Si deseas citar este trabajo, la entrada completa en BibTeX es la siguiente:

```
@mastersthesis{citekey,  
    title = {Desarrollo de una aplicación en Python para el estudio y programación  
de SmartCards SLE5528 y SLE5542},  
    author = {David Balenzategui García y José Gutiérrez Fernández},  
    school = {E.T.S. de Ingeniería de Sistemas Informáticos},  
    year = {2025},  
    month = {11},  
    type = {Trabajo Fin de Grado}  
}
```

Esta obra está bajo una licencia [Creative Commons «Atribución-NoComercial-CompartirIgual 4.0 Internacional»](#). Obra derivada de <https://github.com/blazaid/UPM-Report-Template>.



Todo cambio respecto a la obra original es responsabilidad exclusiva del presente autor.

Agradecimientos

Este proyecto ha sido posible gracias al apoyo y colaboración de varias personas a las que deseo expresar mi más sincero agradecimiento.

En primer lugar, quiero agradecer a mi tutor, José Gutiérrez, por su guía, paciencia y valiosos consejos durante todo el desarrollo de este Trabajo de Fin de Grado. Sus orientaciones han sido fundamentales para alcanzar los objetivos propuestos.

A la Escuela Técnica Superior de Ingeniería de Sistemas Informáticos (ETSIISI) de la Universidad Politécnica de Madrid, por proporcionar los recursos necesarios y el entorno académico que ha permitido mi formación.

A mis compañeros de carrera, por el apoyo mutuo, las largas horas de estudio compartidas y los momentos de ánimo en los días más difíciles.

A mi familia y mi pareja, por su apoyo incondicional, su comprensión durante las largas jornadas de trabajo y por creer siempre en mí. Este logro también es suyo.

A todos aquellos que, de una u otra forma, han contribuido a que este proyecto llegue a buen puerto.

Gracias.

Resumen

La asignatura de Comercio Electrónico de la ETSISI incluye el estudio y programación de SmartCards, dispositivos que incorporan mecanismos de protección complejos cuyo desconocimiento puede provocar bloqueos permanentes. Este proyecto desarrolla CardSIM, una aplicación de escritorio en Python que facilita el aprendizaje de los mecanismos de funcionamiento y protección de SmartCards SLE5528 y SLE5542.

La herramienta proporciona una vista completa y estructurada del contenido de las tarjetas con sus diferentes zonas de memoria y privilegios de acceso, permite trabajar simultáneamente con múltiples tarjetas virtuales antes de su programación física. El sistema implementa una máquina de estados que replica fielmente el comportamiento de las tarjetas reales, incluyendo sus mecanismos de seguridad basados en PSC (Programmable Security Code), contador de errores y bits de protección permanente.

La interfaz gráfica ofrece un panel de comandos APDU interactivo con validación en tiempo real y un sistema de registro completo de operaciones que facilita la depuración y el aprendizaje. La aplicación utiliza el lector ACR39U y permite tanto la simulación completa como la lectura y escritura de tarjetas físicas, habilitando un flujo de trabajo educativo seguro donde los estudiantes pueden experimentar libremente en un entorno virtual antes de trabajar con hardware real.

CardSIM está diseñada para minimizar los bloqueos de tarjetas mediante validaciones preventivas, advertencias claras sobre operaciones irreversibles y confirmaciones obligatorias en acciones críticas. Esto reduce significativamente los costos educativos y mejora la experiencia de aprendizaje de los estudiantes en el manejo de tarjetas inteligentes.

Palabras clave: SmartCards; SLE5528; SLE5542; Python; Comercio Electrónico; Educación; Simulador de tarjetas

Abstract

The Electronic Commerce course at ETSISI includes the study and programming of SmartCards, devices that incorporate complex protection mechanisms whose misunderstanding can cause permanent blocks. This project develops CardSIM, a Python desktop application that facilitates learning the functioning and protection mechanisms of SLE5528 and SLE5542 SmartCards.

The tool provides a complete and structured view of card contents with their different memory zones and access privileges, allows simultaneous work with multiple virtual cards before physical programming. The system implements a state machine that faithfully replicates the behavior of real cards, including their security mechanisms based on PSC (Programmable Security Code), error counter and permanent protection bits.

The graphical interface offers an interactive APDU command panel with real-time validation and a complete operation logging system that facilitates debugging and learning. The application uses the ACR39U reader and supports both complete simulation and physical card reading/writing, enabling a safe educational workflow where students can freely experiment in a virtual environment before working with real hardware.

CardSIM is designed to minimize card blocks through preventive validations, clear warnings about irreversible operations and mandatory confirmations on critical actions. This significantly reduces educational costs and improves students' learning experience in handling smart card technology.

Keywords: SmartCards; SLE5528; SLE5542; Python; Electronic Commerce; Education; Card Simulator

Índice general

| | |
|---|-----|
| Agradecimientos | I |
| Resumen | II |
| Abstract | III |
| Índice de figuras | VI |
| Índice de tablas | XI |
| Índice de Siglas | XIV |
| 1 Introducción | 1 |
| 1.1 Contexto del proyecto | 2 |
| 1.2 Motivación | 2 |
| 1.3 Aspectos sociales, éticos, ambientales, legales y económicos .. | 4 |
| 1.4 Contribución a los Objetivos de Desarrollo Sostenible (ODS) .. | 6 |
| 2 Objetivos | 7 |
| 2.1 Objetivo general | 7 |
| 2.2 Objetivos específicos | 7 |
| 3 Medios disponibles | 8 |

| | |
|---|------------|
| 3.1 Recursos hardware | 8 |
| 3.2 Recursos software | 9 |
| 3.3 Recursos bibliográficos | 14 |
| 4 Gestión del proyecto | 16 |
| 4.1 Metodología de trabajo | 16 |
| 4.2 Análisis de requisitos | 17 |
| 4.3 Planificación temporal | 24 |
| 5 Marco teórico | 29 |
| 5.1 Tarjetas inteligentes SLE5528 y SLE5542 | 29 |
| 5.2 Protocolo APDU (ISO 7816-4) | 29 |
| 5.3 Características de la tarjeta SLE5542 | 33 |
| 5.4 Características de la tarjeta SLE5528 | 34 |
| 5.5 Comandos APDU para tarjetas de memoria | 35 |
| 6 Sistema desarrollado | 45 |
| 6.1 Arquitectura del sistema | 45 |
| 6.2 Módulos principales | 47 |
| 6.3 Interfaz de usuario integrada | 48 |
| 6.4 Flujos de trabajo típicos | 104 |
| 7 Resultados, Conclusiones y Líneas Futuras | 107 |
| 7.1 Resultados | 107 |
| 7.2 Conclusiones | 110 |

| | |
|---|------------|
| 7.3 Líneas futuras de trabajo | 115 |
| 8 Bibliografía | 118 |
| 9 Anexos | 120 |
| 9.1 Anexo A: Estructura del proyecto | 120 |
| 9.2 Anexo B: Ejemplo tarjeta virtual guardada en .txt | 121 |
| 9.3 Anexo C: Códigos de respuesta (Status Words) | 122 |
| 9.4 Anexo D: Instalación y configuración | 123 |
| 9.5 Anexo E: Atajos de teclado | 124 |
| 9.6 Anexo F: Esquema de colores de memoria | 124 |

Índice de figuras

| | | |
|-----|---|----|
| 1.1 | Interfaz de c3tool: terminal de consola con respuestas poco claras | 3 |
| 1.2 | CardSim antiguo: funcionalidad limitada a tarjetas SLE5542 de 256 bytes | 4 |
| 1.3 | ODS 4: Educación de calidad | 6 |
| 1.4 | ODS 9: Industria, innovación e infraestructura | 6 |
| 1.5 | ODS 12: Producción y consumo responsables | 6 |
| 3.1 | ACR39U (modelo H1, color blanco) | 8 |
| 3.2 | ACR39U (modelo UF, color negro) | 8 |
| 3.3 | Ejemplos de tarjetas SLE5542 y SLE5528 utilizadas en prácticas | 9 |
| 3.4 | Logo de Python | 10 |
| 3.5 | Logo de Visual Studio Code | 13 |
| 3.6 | Logo de Git | 13 |
| 3.7 | Logo de GitHub | 13 |
| 3.8 | Logo de LaTeX | 14 |
| 4.1 | Diagrama de Gantt del proyecto | 27 |
| 5.1 | Microprocesador de una tarjeta de contacto | 31 |
| 6.1 | Vista inicial de CardSIM mostrando la distribución completa de paneles | 48 |

| | |
|---|----|
| 6.2 Vista de CardSIM con tarjetas cargadas y operaciones en curso | 50 |
| 6.3 Panel Commands con botones de gestión de archivos | 52 |
| 6.4 Diálogo creación SLE5542 | 53 |
| 6.5 Diálogo creación SLE5528 | 53 |
| 6.6 Diálogo para abrir tarjetas previamente guardadas | 54 |
| 6.7 Diálogo para guardar tarjetas en archivo | 55 |
| 6.8 Confirmación antes de cerrar una tarjeta | 56 |
| 6.9 Confirmación antes de restaurar tarjeta a estado de fábrica | 57 |
| 6.10 Diálogo de configuración de usuario | 58 |
| 6.11 Estado inicial - Solo SELECT CARD TYPE habilitado | 60 |
| 6.12 Despues de SELECT - Comandos de lectura habilitados | 61 |
| 6.13 Despues de PRESENT CODE - SLE5542 con escritura habilitada | 61 |
| 6.14 Despues de PRESENT CODE - SLE5528 con escritura habilitada | 61 |
| 6.15 Diálogo SELECT CARD TYPE | 62 |
| 6.16 Diálogo READ MEMORY para SLE5542 | 63 |
| 6.17 Diálogo READ MEMORY para SLE5528 | 63 |
| 6.18 Confirmación exitosa de lectura con datos en hexadecimal | 64 |
| 6.19 Error al intentar leer fuera del rango válido | 64 |
| 6.20 Diálogo PRESENT CODE para SLE5542 (PSC de 3 bytes) | 65 |
| 6.21 Diálogo PRESENT CODE para SLE5528 (PSC de 2 bytes) | 65 |
| 6.22 Confirmación de presentación correcta del PSC | 66 |
| 6.23 Diálogo WRITE MEMORY para SLE5542 | 67 |
| 6.24 Diálogo WRITE MEMORY para SLE5528 | 67 |

| | |
|---|----|
| 6.25 Confirmación de escritura exitosa | 68 |
| 6.26 Diálogo CHANGE PSC para SLE5542 | 69 |
| 6.27 Diálogo CHANGE PSC para SLE5528 | 69 |
| 6.28 Advertencia sobre el cambio de PSC | 69 |
| 6.29 Resultado del comando READ ERROR COUNTER | 70 |
| 6.30 Confirmación de lectura de Protection Bits | 71 |
| 6.31 Resultado visual de Protection Bits para SLE5542 | 72 |
| 6.32 Resultado visual de Protection Bits para SLE5528 | 72 |
| 6.33 Diálogo WRITE PROTECT para SLE5542 | 74 |
| 6.34 Diálogo WRITE PROTECT para SLE5528 | 74 |
| 6.35 Intento de protección sin coincidencia de datos | 75 |
| 6.36 Protección exitosa de direcciones | 75 |
| 6.37 Panel de información para tarjeta SLE5542 | 76 |
| 6.38 Panel de información para tarjeta SLE5528 | 76 |
| 6.39 Panel con múltiples tarjetas abiertas | 77 |
| 6.40 Panel después de cerrar algunas tarjetas | 78 |
| 6.41 Visualización de memoria para tarjeta SLE5542 (256 bytes) | 79 |
| 6.42 Visualización de memoria para tarjeta SLE5528 (1024 bytes - Página 3 activa) | 80 |
| 6.43 Log mostrando inicialización de la aplicación | 83 |
| 6.44 Log mostrando cambio de PSC, lectura de memoria y consulta del contador de errores | 84 |
| 6.45 Log con secuencia de escritura, protección y error de presentación de PSC | 85 |

| | |
|---|-----|
| 6.46 Log mostrando bloqueo permanente de tarjeta por intentos erróneos consecutivos | 85 |
| 6.47 Confirmación para resetear el contador de errores | 86 |
| 6.48 Panel de operaciones con tarjetas físicas | 88 |
| 6.49 Diálogo de lectura desde tarjeta física | 89 |
| 6.50 Resultado de lectura con PSC por defecto (SLE5528) | 90 |
| 6.51 Opción para introducir PSC personalizado | 91 |
| 6.52 Creación de nueva tarjeta virtual desde lectura física | 91 |
| 6.53 Diálogo de escritura a tarjeta física SLE5542 | 92 |
| 6.54 Diálogo de escritura a tarjeta física SLE5528 | 92 |
| 6.55 Panel Actions con botones de funcionalidades adicionales | 94 |
| 6.56 Diálogo para guardar el log de comandos | 95 |
| 6.57 Confirmación de guardado exitoso del log | 95 |
| 6.58 Confirmación antes de limpiar el log | 95 |
| 6.59 Ventana de referencia de comandos APDU | 96 |
| 6.60 Ventana de créditos y reconocimientos | 97 |
| 6.61 Configuración general de la aplicación | 98 |
| 6.62 Activación del modo para pantallas pequeñas desde Settings | 99 |
| 6.63 Interfaz adaptada a pantallas pequeñas | 99 |
| 6.64 Panel de tarjetas abiertas en modo compacto | 100 |
| 6.65 Confirmación de activación del modo administrador | 101 |
| 6.66 Panel con funcionalidades administrativas visibles | 102 |
| 6.67 Ventana de cambio de PSC para tarjetas físicas para SLE5542 | 103 |

| | |
|--|-----|
| 6.68 Ventana de cambio de PSC para tarjetas físicas para SLE5528 . . . | 103 |
| 6.69 Log mostrando desbloqueo de modo administrador, cambios en configuración de GUI y error de acceso al lector | 104 |

Índice de tablas

| | | |
|------|---|----|
| 4.1 | Priorización de requisitos de CardSIM | 24 |
| 5.1 | Estructura del mensaje C-APDU | 32 |
| 5.2 | Estructura del mensaje R-APDU | 32 |
| 5.3 | Comando SELECT_CARD_TYPE para SLE5542 | 35 |
| 5.4 | Comando SELECT_CARD_TYPE para SLE5528 | 36 |
| 5.5 | Respuesta exitosa SELECT_CARD_TYPE | 36 |
| 5.6 | Comando READ_MEMORY_CARD para SLE5542 | 36 |
| 5.7 | Comando READ_MEMORY_CARD para SLE5528 | 36 |
| 5.8 | Respuesta READ_MEMORY_CARD SLE5542 | 37 |
| 5.9 | Respuesta READ_MEMORY_CARD SLE5528 | 37 |
| 5.10 | Comando PRESENT_CODE para SLE5542 | 37 |
| 5.11 | Comando PRESENT_CODE para SLE5528 | 38 |
| 5.12 | Respuesta PRESENT_CODE SLE5542 | 38 |
| 5.13 | Respuesta PRESENT_CODE SLE5528 | 38 |
| 5.14 | Comando WRITE_MEMORY_CARD para SLE5542 | 39 |
| 5.15 | Comando WRITE_MEMORY_CARD para SLE5528 | 39 |
| 5.16 | Respuesta WRITE_MEMORY_CARD | 39 |
| 5.17 | Comando READ_ERROR_COUNTER para SLE5542 | 39 |

| | |
|---|-----|
| 5.18 Respuesta READ_ERROR_COUNTER SLE5542 | 40 |
| 5.19 Comando READ_ERROR_COUNTER para SLE5528 | 40 |
| 5.20 Respuesta READ_ERROR_COUNTER SLE5528 | 40 |
| 5.21 Comando READ_PROTECTION_BITS para SLE5542 | 41 |
| 5.22 Respuesta READ_PROTECTION_BITS SLE5542 | 41 |
| 5.23 Comando READ_PROTECTION_BITS para SLE5528 | 41 |
| 5.24 Respuesta READ_PROTECTION_BITS SLE5528 | 41 |
| 5.25 Comando WRITE_PROTECT para SLE5542 (solo @00h-1Fh) | 42 |
| 5.26 Comando WRITE_PROTECT para SLE5528 | 42 |
| 5.27 Respuesta WRITE_PROTECT | 43 |
| 5.28 Comando CHANGE_CODE para SLE5542 | 43 |
| 5.29 Cambio de PSC en SLE5528 mediante WRITE_MEMORY | 43 |
| 5.30 Respuesta CHANGE_CODE | 44 |
| 6.1 Tecnologías utilizadas en CardSIM | 46 |
| 9.1 Status Words implementados | 122 |
| 9.2 Atajos de teclado de CardSIM | 124 |
| 9.3 Esquema de colores de visualización | 124 |

Índice de Siglas

APDU - Application Protocol Data Unit. Unidad de datos del protocolo de aplicación definido por ISO 7816-4 para comunicación con tarjetas inteligentes.

ASCII - American Standard Code for Information Interchange. Código estándar para representación de caracteres en formato texto.

ATR - Answer To Reset. Respuesta al reinicio que una tarjeta inteligente envía al lector tras ser insertada, conteniendo información sobre sus capacidades.

CLA - Class Byte. Primer byte de un comando APDU que identifica la clase de instrucción según ISO 7816-4.

CLI - Command Line Interface. Interfaz de línea de comandos para interacción con sistemas mediante texto.

TFG - Trabajo de Fin de Grado. Proyecto académico final requerido para obtener el título de grado.

UPM - Universidad Politécnica de Madrid. Institución académica donde se desarrolla el proyecto.

ETSI SI - Escuela Técnica Superior de Ingeniería de Sistemas Informáticos. Centro de la Universidad Politécnica de Madrid donde se imparte la asignatura de Comercio Electrónico.

GUI - Graphical User Interface. Interfaz gráfica de usuario que permite interacción visual con la aplicación.

IDE - Integrated Development Environment. Entorno de desarrollo integrado.

INS - Instruction Byte. Segundo byte de un comando APDU que especifica la instrucción concreta a ejecutar.

ISO - International Organization for Standardization. Organización Internacional de Normalización que establece estándares técnicos globales.

JSON - JavaScript Object Notation. Formato ligero de intercambio de datos.

P1, P2 - Parameter Bytes 1 y 2. Tercer y cuarto bytes de un comando APDU que especifican parámetros adicionales de la instrucción.

PC/SC - Personal Computer/Smart Card. Especificación estándar para comunicación entre ordenadores personales y lectores de tarjetas inteligentes.

PSC - Programmable Security Code. Código de seguridad programable de las tarjetas SLE5528 y SLE5542 que protege el acceso a ciertas operaciones.

SW1, SW2 - Status Word Bytes 1 y 2. Bytes de estado que una tarjeta devuelve en respuesta a un comando APDU indicando el resultado de la operación.

1.

Introducción

El desarrollo de aplicaciones basadas en SmartCards constituye un área fundamental dentro de la seguridad informática y del comercio electrónico. Estos dispositivos, con apariencia de tarjeta de crédito pero con un microprocesador y memoria incorporados, permiten la gestión segura de datos, autenticación de usuarios y ejecución de transacciones digitales. Su presencia en la vida cotidiana abarca desde los sistemas de pago hasta la identificación personal, el control de accesos y la provisión de servicios gubernamentales.

En la Escuela Técnica Superior de Ingeniería de Sistemas Informáticos (ETSI-SI) de la Universidad Politécnica de Madrid (UPM), la asignatura de Comercio Electrónico dedica parte de sus contenidos al estudio y programación de estas tecnologías, en particular de las tarjetas SLE5528 y SLE5542. No obstante, su aprendizaje presenta varios desafíos: el desconocimiento de los mecanismos de protección y de los códigos de seguridad puede llevar fácilmente al bloqueo permanente de la tarjeta, dejándola inservible y generando tanto pérdidas económicas como una limitación práctica en la docencia.

Cada curso, con una media de 20–30 estudiantes matriculados, se utilizan entre 10 y 20 tarjetas físicas, de las cuales entre 3 y 4 suelen quedar bloqueadas. Este hecho supone un coste recurrente para la asignatura, reduce la libertad del alumnado para experimentar y genera un ambiente de presión que puede impactar negativamente en el aprendizaje. Además, la escasa documentación técnica sobre estas tarjetas, unida a que la herramienta actualmente utilizada en la asignatura (c3tool) ofrece una interfaz limitada y poca retroalimentación, hace que la curva de aprendizaje sea más lenta y compleja.

Con el fin de responder a esta problemática, este Trabajo Fin de Grado presenta el proyecto “Desarrollo de una aplicación en Python para el estudio y programación de SmartCards SLE5528 y SLE5542”, cuyo resultado principal es la aplicación de escritorio CardSIM. Esta herramienta está diseñada para facilitar el aprendizaje práctico mediante un entorno controlado y seguro, reduciendo riesgos de bloqueo y mejorando la experiencia formativa.

1.1. Contexto del proyecto

El uso de tarjetas inteligentes en entornos educativos requiere un equilibrio entre el aprendizaje práctico y la preservación de recursos físicos. La falta de documentación accesible y la complejidad técnica dificultan la labor de los estudiantes, quienes muchas veces se ven limitados a realizar pruebas conservadoras para evitar bloqueos irreversibles.

El desarrollo de CardSIM busca superar estas limitaciones y ofrecer:

- **Entorno de simulación con múltiples tarjetas virtuales:** permite a los alumnos enviar comandos y APDUs antes de trabajar con tarjetas reales.
- **Lectura de tarjetas físicas:** mediante un lector ACR39U y su carga en el entorno virtual.
- **Escritura en tarjetas reales:** posibilidad de transferir el contenido desde la simulación hacia una tarjeta física.
- **Guardado y carga de tarjetas virtuales:** opción para retomar prácticas previas sin depender exclusivamente del material físico.

Gracias a ello, el alumnado puede experimentar con mayor libertad, adquirir confianza y reducir los errores que conducen a bloqueos definitivos.

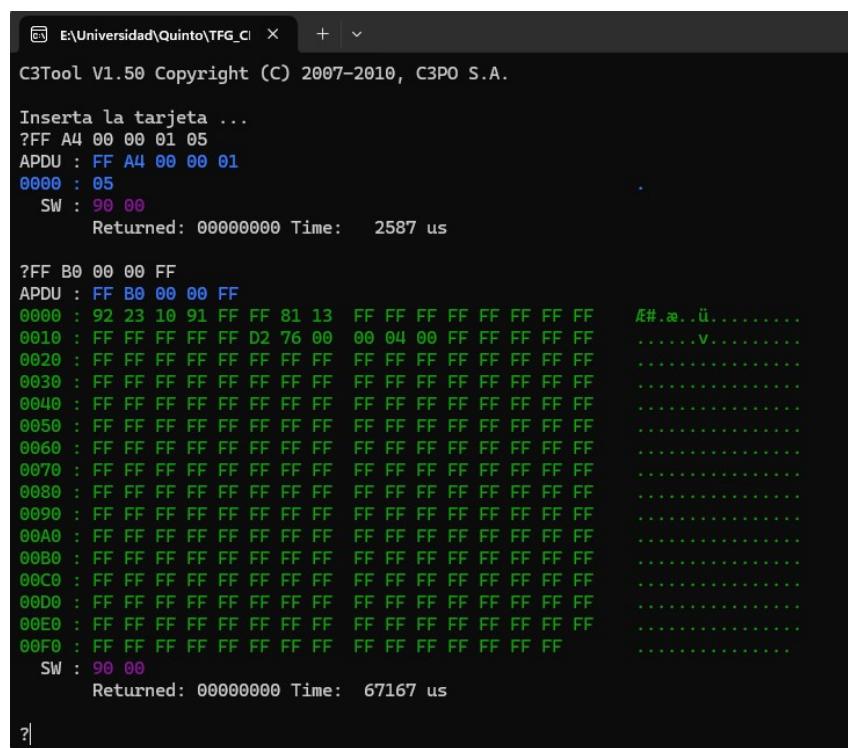
1.2. Motivación

La motivación principal de este proyecto surge de la observación directa de las dificultades que afrontan los estudiantes en el laboratorio. Los problemas más destacados son:

- **Bloqueos frecuentes:** cada curso se pierden varias tarjetas, lo que representa un porcentaje significativo del material disponible.

- **Curva de aprendizaje lenta:** el alumnado necesita varias sesiones para familiarizarse con los comandos básicos y los mecanismos de protección.
- **Carencia de herramientas educativas:** las existentes, como c3tool, se limitan a transmitir APDUs sin proporcionar un feedback claro.
- **Escasa documentación técnica:** los manuales disponibles suelen estar orientados a desarrolladores de bajo nivel, no a estudiantes en formación.

Hasta el momento, en la asignatura se han utilizado principalmente dos herramientas con importantes limitaciones. Por un lado, **c3tool** es una aplicación de consola que no muestra de manera clara el funcionamiento interno de las operaciones, devolviendo respuestas poco intuitivas y dificultando la comprensión del alumnado.



The screenshot shows a terminal window titled 'E:\Universidad\Quinto\TFG_CI' running 'C3Tool V1.50 Copyright (C) 2007-2010, C3PO S.A.'. The user has entered a command to insert a card, followed by an APDU (A4 00 00 01 05). The tool returns the APDU (FF A4 00 00 01), the SW value (0000 : 05), and a timestamp (Returned: 00000000 Time: 2587 us). The next command issued is '?FF B0 00 00 FF', which returns a large amount of binary data representing the card's memory dump. The dump consists of 256 bytes of FF followed by a sequence of 00 values, with a timestamp (Returned: 00000000 Time: 67167 us) at the end. The terminal prompt '?' is visible at the bottom.

Figura 1.1. Interfaz de c3tool: terminal de consola con respuestas poco claras

Por otro lado, existía un **simulador antiguo** (CardSim) que únicamente permitía trabajar con tarjetas de 256 bytes (SLE5542) y ofrecía funciones muy limitadas, sin soporte para el modelo SLE5528 ni para operaciones avanzadas.

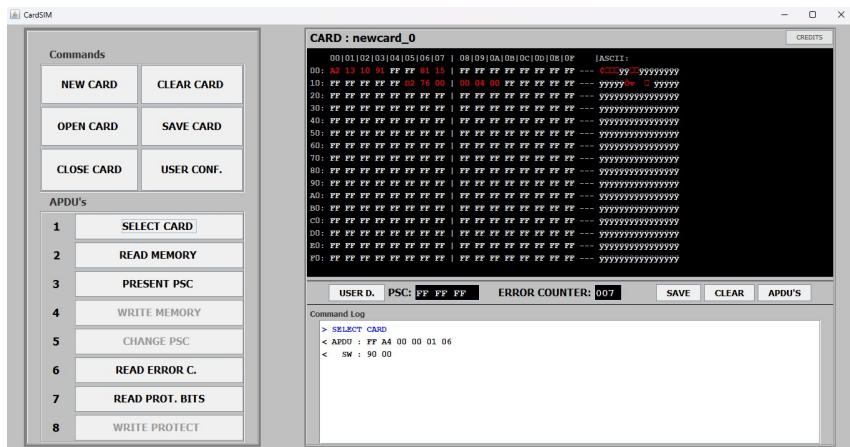


Figura 1.2. CardSim antiguo: funcionalidad limitada a tarjetas SLE5542 de 256 bytes

Nota sobre nomenclatura: El nuevo simulador desarrollado en este proyecto ha reutilizado el nombre "CardSIM" del simulador antiguo, dada su naturaleza como sucesor y reemplazo directo de la herramienta anterior. Ambas aplicaciones comparten el propósito educativo de simular tarjetas inteligentes, pero la nueva versión representa una reescritura completa con arquitectura moderna, funcionalidades ampliadas y soporte para ambos tipos de tarjetas (SLE5542 y SLE5528).

Ante estas limitaciones identificadas en las herramientas existentes, la creación de CardSIM pretende ofrecer un recurso pedagógico que combine facilidad de uso, retroalimentación clara y un enfoque preventivo frente a los errores más habituales.

1.3. Aspectos sociales, éticos, ambientales, legales y económicos

Este proyecto no solo aborda aspectos técnicos y educativos, sino que también tiene impactos positivos en varias dimensiones importantes que conviene analizar: responsabilidad social, sostenibilidad y cumplimiento normativo.

Aspecto social. El proyecto facilita el acceso a tecnologías de seguridad para todos los estudiantes, permitiéndoles aprender en un entorno más flexible y

sin la presión económica de poder bloquear material de trabajo. Así, los estudiantes que necesitan más tiempo para comprender los conceptos pueden practicar libremente sin temor a causar daños irreparables.

Aspecto ético. La aplicación se ha diseñado siguiendo principios de transparencia y responsabilidad. Incluye una funcionalidad opcional de información de usuario (User Info) que permite a cada estudiante registrar voluntariamente su nombre, apellidos, año académico y número de matrícula junto con sus tarjetas virtuales. Estos datos se almacenan exclusivamente en local, en el equipo del propio usuario, garantizando así la privacidad y el control total sobre la información personal.

Aspecto ambiental. La reducción del número de tarjetas físicas bloqueadas y desechadas tiene un impacto positivo en la generación de residuos electrónicos. Al prolongar la vida útil del material disponible y minimizar el desperdicio, el proyecto promueve un uso más sostenible y responsable de los recursos educativos.

Aspecto legal. La aplicación se distribuye como un ejecutable portable desarrollado en Python, utilizando exclusivamente librerías de software libre. El almacenamiento local de toda la información cumple con las directrices del Reglamento General de Protección de Datos (RGPD), ya que no se produce ninguna transferencia de datos personales fuera del equipo del usuario ni se realiza ningún tipo de procesamiento en servidores externos.

Aspecto económico. La reducción del número de tarjetas bloqueadas representa un ahorro directo y tangible para la institución. Cada tarjeta que se evita reemplazar supone un ahorro económico que, multiplicado por el número de estudiantes a lo largo de varios cursos académicos, resulta significativo. Además, el tiempo de práctica en el aula se aprovecha de manera más eficiente, minimizando interrupciones causadas por bloqueos y optimizando la experiencia docente.

1.4. Contribución a los Objetivos de Desarrollo Sostenible (ODS)

Este proyecto también contribuye a la Agenda 2030 a través de varios Objetivos de Desarrollo Sostenible:

ODS 4: Educación de calidad. Favorece el aprendizaje práctico y accesible en materia de seguridad digital, mejorando la formación de los estudiantes en comercio electrónico y ciberseguridad.

ODS 9: Industria, innovación e infraestructura. Impulsa la innovación tecnológica en el ámbito educativo y fomenta infraestructuras digitales seguras, esenciales para el desarrollo de la economía digital.

ODS 12: Producción y consumo responsables. Reduce el desperdicio de recursos físicos y promueve un uso más responsable de las tarjetas inteligentes, al tiempo que genera confianza en transacciones digitales seguras.



Figura 1.3. ODS 4: Educación de calidad



Figura 1.4. ODS 9: Industria, innovación e infraestructura



Figura 1.5. ODS 12: Producción y consumo responsables

2.

Objetivos

Este capítulo define los objetivos del proyecto, estableciendo las metas específicas que se pretenden alcanzar con el desarrollo de la aplicación para el estudio y programación de SmartCards SLE5528 y SLE5542.

2.1. Objetivo general

Desarrollar una aplicación de escritorio en Python que facilite el aprendizaje y la programación segura de SmartCards SLE5528 y SLE5542, minimizando el riesgo de bloqueos permanentes y mejorando significativamente la experiencia educativa en la asignatura de Comercio Electrónico de la ETSISI.

2.2. Objetivos específicos

1. **Diseñar una interfaz gráfica intuitiva y accesible** que permita a los estudiantes interactuar con las SmartCards de manera sencilla y segura, sin necesidad de conocimientos avanzados.
2. **Implementar funcionalidades de lectura y escritura segura** para las tarjetas SLE5528 y SLE5542, incluyendo mecanismos de validación que prevengan operaciones que puedan causar bloqueos permanentes o corrupción de datos.
3. **Integrar el soporte completo para el lector ACR39U**, asegurando la compatibilidad y comunicación efectiva entre la aplicación y el hardware disponible en la ETSISI.
4. **Crear herramientas de visualización de memoria** que faciliten la comprensión de la estructura y organización de datos en las tarjetas, contribuyendo al aprendizaje conceptual y la depuración de problemas.

3.

Medios disponibles

Este capítulo describe los recursos disponibles para el desarrollo del proyecto, incluyendo hardware, software y documentación.

3.1. Recursos hardware

3.1.1. Lector de tarjetas ACR39U

El lector ACR39U constituye el hardware principal para la comunicación con las SmartCards. Este dispositivo cumple con los estándares ISO 7816 y es compatible con las especificaciones PC/SC, proporcionando la interfaz necesaria para la comunicación entre la aplicación y las tarjetas.



Figura 3.1. ACR39U (modelo H1, color blanco)



Figura 3.2. ACR39U (modelo UF, color negro)

3.1.2. Tarjetas SmartCard



Figura 3.3. Ejemplos de tarjetas SLE5542 y SLE5528 utilizadas en prácticas

- **SLE5528:** Tarjetas de memoria de 1024 bytes
- **SLE5542:** Tarjetas de memoria de 256 bytes

3.2. Recursos software

3.2.1. Lenguaje de programación

- **Python 3.11:** Lenguaje principal de desarrollo seleccionado por múltiples razones estratégicas:
 - *Compatibilidad con librería pyscard:* Razón fundamental para la elección de Python. Esta biblioteca especializada permite la comunicación con lectores de tarjetas a través del estándar PC/SC.

- *Legibilidad y mantenibilidad:* Sintaxis clara y expresiva que facilita el mantenimiento futuro del código por parte de estudiantes y docentes sin experiencia avanzada en programación.
- *Multiplataforma:* Garantiza la portabilidad de la aplicación entre diferentes sistemas operativos (Windows, Linux, macOS) sin modificaciones significativas en el código.
- *Ecosistema de bibliotecas:* Amplia disponibilidad de librerías para diversas funcionalidades (interfaz gráfica, serialización de datos, logging, etc.).
- *Documentación y comunidad:* Extensa documentación oficial y gran comunidad activa que facilita la resolución de problemas y el aprendizaje continuo.



Figura 3.4. Logo de Python

3.2.2. Librerías utilizadas

Comunicación con hardware

- **pyscard:** Biblioteca especializada para la comunicación con SmartCards a través del estándar PC/SC (Personal Computer/Smart Card). Proporciona una abstracción de alto nivel sobre los lectores de tarjetas, permitiendo enviar y recibir comandos APDU de forma programática. Es la biblioteca fundamental sobre la que se construye toda la funcionalidad de comunicación física.
- **smartcard:** Módulo complementario de pyscard que ofrece funcionalidades de bajo nivel para la comunicación directa con las tarjetas. Proporciona las clases `System.readers()` para detectar lectores conectados, gestión de conexiones con tarjetas físicas, y utilidades como `toHexString()` y `toBytes()` para conversión de formatos de datos.

Interfaz gráfica y componentes visuales

- **tkinter:** Biblioteca estándar de Python para el desarrollo de interfaces gráficas. Se eligió Tkinter frente a alternativas más modernas como PyQt o wxPython por varios motivos estratégicos:
 - *Inclusión en distribución estándar:* Viene preinstalado con Python, eliminando dependencias externas complejas.
 - *Curva de aprendizaje:* Su simplicidad facilita el mantenimiento futuro del código por estudiantes o docentes sin experiencia previa.
 - *Madurez y estabilidad:* Amplia trayectoria que garantiza comportamiento consistente en todas las plataformas.
 - *Rendimiento adecuado:* Suficiente para las necesidades de la aplicación sin la sobrecarga de frameworks más pesados.
 - *Documentación extensa:* Amplia disponibilidad de recursos educativos, ideal para un contexto académico.

Submódulos utilizados:

- **messagebox:** Cuadros de diálogo para mensajes de información, advertencia y confirmación.
- **scrolledtext:** Widgets de texto con scroll integrado, utilizados para el panel de log de comandos.
- **filedialog:** Diálogos para selección de archivos (abrir/guardar tarjetas virtuales).
- **simpledialog:** Diálogos simples para entrada de datos del usuario.
- **PIL (Pillow):** Biblioteca de procesamiento de imágenes. Utilizada para cargar, redimensionar y convertir iconos e imágenes de la interfaz gráfica, incluyendo el ícono de la ETSISI.

Gestión de datos y serialización

- **json:** Módulo estándar para serialización y deserialización de datos. Utilizado para:

- Guardar configuración persistente del usuario (User Info) en el archivo `user_config.json`
- Crear archivos temporales de sesión para cada tarjeta virtual abierta, almacenando el estado completo de la sesión (PSC verificado, contador de errores, estado de bloqueo, etc.)
- Gestión de metadatos y configuración de la aplicación

Nota: Las tarjetas virtuales se guardan en formato `.txt` con estructura de texto plano para facilitar la lectura y edición manual por parte de los usuarios. El formato JSON se reserva para configuración y archivos temporales de sesión.

- **uuid:** Generación de identificadores únicos universales. Cada sesión de tarjeta virtual recibe un UUID único para gestión interna y trazabilidad.

Gestión de fechas y tiempos

- **datetime:** Módulo para gestión de fechas y timestamps. Utilizado para:
 - Registro temporal de todas las operaciones en el log de comandos
 - Marcas de tiempo de creación de tarjetas virtuales
 - Trazabilidad cronológica de sesiones de trabajo

Gestión de archivos y sistema

- **os:** Interacción con el sistema operativo. Utilizado para gestión de rutas de archivos, detección de recursos, y compatibilidad multiplataforma.
- **sys:** Acceso a parámetros y funciones del intérprete Python. Empleado para gestión de paths de importación y control del ciclo de vida de la aplicación.
- **tempfile:** Creación de archivos y directorios temporales seguros. Utilizado para operaciones intermedias que no requieren persistencia.

3.2.3. Herramientas de desarrollo

Entornos de desarrollo integrados (IDEs)

- **Visual Studio Code:** Editor de código principal utilizado para el desarrollo de la aplicación Python. Proporciona extensiones especializadas para Python, depuración integrada, control de versiones Git incorporado, y terminal integrada.



Figura 3.5. Logo de Visual Studio Code

Control de versiones y colaboración

- **Git:** Sistema de control de versiones distribuido para el seguimiento de cambios en el código fuente. Permite mantener un historial completo de modificaciones y facilita la reversión de cambios cuando es necesario.
- **GitHub:** Plataforma de alojamiento remoto del repositorio Git. Proporciona respaldo en la nube del código fuente y documentación del proyecto.



Figura 3.6. Logo de Git

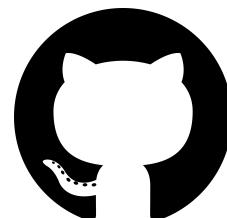


Figura 3.7. Logo de GitHub

Empaquetado y distribución

- **PyInstaller:** Herramienta para la generación de ejecutables a partir del código Python. Permite distribuir la aplicación como un archivo ejecutable (.exe en Windows) que incluye el intérprete de Python y todas las dependencias, simplificando la utilización de la aplicación para los alumnos de Comercio Electrónico.

Documentación y redacción académica

- **LaTeX:** Sistema de composición de documentos de alta calidad tipográfica. Utilizado para la redacción de la memoria del proyecto, garantizando formato profesional y gestión eficiente de referencias bibliográficas.
- **TeX Live / MiKTeX:** Distribución completa de LaTeX que incluye el compilador y todos los paquetes necesarios para la compilación de documentos.
- **Visual Studio Code:** Editor también utilizado para la edición de archivos LaTeX.



Figura 3.8. Logo de LaTeX

3.3. Recursos bibliográficos

3.3.1. Documentación técnica

- Especificaciones técnicas SLE5528 y SLE5542 de Infineon
- Manual de programación del lector ACR39U

- Estándares ISO 7816 para SmartCards
- Especificaciones PC/SC para comunicación con lectores

3.3.2. Literatura académica

- Artículos sobre seguridad en SmartCards
- Documentación sobre mejores prácticas en desarrollo de software educativo

4.

Gestión del proyecto

Este capítulo describe la metodología empleada para el desarrollo del proyecto, así como los aspectos de planificación temporal del mismo.

4.1. Metodología de trabajo

4.1.1. Enfoque metodológico

Se ha adoptado un enfoque iterativo e incremental basado en los siguientes principios:

- **Desarrollo centrado en el usuario:** Comunicación continua con el tutor de la asignatura para asegurar que la herramienta cumple con las necesidades educativas.
- **Prototipado rápido:** Desarrollo de prototipos funcionales en etapas tempranas para validar conceptos y obtener retroalimentación.
- **Mejora continua:** Incorporación de mejoras basadas en pruebas y comentarios recibidos durante el desarrollo.
- **Documentación evolutiva:** Redacción progresiva de la memoria y documentación técnica a medida que avanza el proyecto.

4.1.2. Fases del proyecto

El proyecto se estructura en cinco fases principales que definen la organización conceptual del trabajo:

1. Análisis y diseño

Definición del problema, requisitos y arquitectura del sistema. Incluye el estudio de las necesidades educativas de la asignatura, análisis de las limitaciones de las herramientas existentes, y diseño de la solución propuesta.

2. Desarrollo de la interfaz gráfica

Implementación de la GUI con Tkinter, diseño de los componentes visuales (vista de memoria, panel de comandos, gestión de tarjetas), y creación de la experiencia de usuario.

3. Desarrollo de funcionalidades core

Implementación de la lógica de la aplica: sistema de tarjetas virtuales, gestión de memoria y protecciones, guardado/carga de tarjetas, validación de comandos APDU, y mecanismos de seguridad.

4. Integración de comunicación con tarjetas físicas

Implementación de la comunicación con lectores ACR39U mediante pyscard, desarrollo de operaciones de lectura/escritura en tarjetas reales, y sincronización entre tarjetas virtuales y físicas.

5. Validación y documentación

Pruebas en entorno real con estudiantes, corrección de errores, optimización de la experiencia de usuario, redacción de la memoria y preparación de la presentación.

4.2. Análisis de requisitos

El análisis de requisitos constituye una fase fundamental para el desarrollo exitoso de CardSIM. Esta sección describe los requisitos funcionales y no funcionales identificados durante la fase de análisis, que sirvieron como guía para el diseño e implementación de la aplicación.

4.2.1. Requisitos funcionales

Los requisitos funcionales definen las capacidades y comportamientos específicos que debe ofrecer CardsIM. Se han organizado en grupos temáticos para facilitar su comprensión y trazabilidad:

Gestión de tarjetas virtuales

- **RF1 - Creación de tarjetas virtuales:** El sistema debe permitir crear tarjetas virtuales de tipo SLE5542 (256 bytes) y SLE5528 (1024 bytes) con configuración de fábrica (memoria inicializada en 0xFF, PSC por defecto, contador de errores en estado inicial).
- **RF2 - Guardado de tarjetas:** El sistema debe permitir exportar tarjetas virtuales a archivos de texto estructurado (formato .txt) que incluyan volcado completo de memoria, PSC actual, estado del contador de errores, configuración de Protection Bits y metadatos de autoría.
- **RF3 - Carga de tarjetas:** El sistema debe permitir importar tarjetas previamente guardadas, detectando automáticamente el tipo de tarjeta y restaurando completamente su estado (memoria, PSC, contador de errores, protecciones).
- **RF4 - Gestión de sesiones múltiples:** El sistema debe soportar hasta 8 tarjetas abiertas simultáneamente, permitiendo cambiar entre ellas mediante un panel de navegación visual.
- **RF5 - Cierre de tarjetas:** El sistema debe permitir cerrar tarjetas individuales, solicitando confirmación al usuario.

Comandos APDU

- **RF6 - SELECT CARD TYPE:** Implementar comando para activar la tarjeta virtual y habilitar operaciones de lectura.
- **RF7 - READ MEMORY:** Implementar lectura de rangos arbitrarios de memoria con validación de direcciones y longitudes según el tipo de tarjeta.

- **RF8 - PRESENT CODE:** Implementar presentación del PSC con verificación byte a byte, control del contador de errores y habilitación de operaciones de escritura tras autenticación exitosa.
- **RF9 - WRITE MEMORY:** Implementar escritura de datos en formato ASCII o hexadecimal, con validación de límites de longitud, verificación de zonas protegidas y requisito de autenticación previa.
- **RF10 - CHANGE PSC:** Implementar cambio del código de seguridad con advertencias al usuario sobre la criticidad de recordar el nuevo código.
- **RF11 - READ ERROR COUNTER:** Implementar lectura del contador de errores con interpretación correcta de las secuencias específicas de cada tipo de tarjeta.
- **RF12 - READ PROTECTION BITS:** Implementar lectura de bits de protección mostrando el estado completo de protección de la memoria.
- **RF13 - WRITE PROTECT:** Implementar protección irreversible de direcciones de memoria con validación de coincidencia de datos y confirmación del usuario.

Máquina de estados

- **RF14 - Control de estados:** El sistema debe implementar una máquina de estados que replique el comportamiento de tarjetas físicas: estado inicial (solo SELECT habilitado) → estado post-SELECT (lectura habilitada) → estado autenticado (escritura habilitada).
- **RF15 - Habilitación dinámica de comandos:** Los botones de comandos APDU deben habilitarse/deshabilitarse automáticamente según el estado actual de la tarjeta.
- **RF16 - Bloqueo por contador de errores:** El sistema debe bloquear permanentemente las operaciones de escritura cuando el contador de errores llega a cero, manteniendo disponibles las operaciones de lectura. El sistema deberá implementar un botón para reiniciar el contador de errores.

Visualización de memoria

- **RF17 - Vista estructurada:** El sistema debe mostrar el contenido de memoria en formato de tres columnas (Dirección, Hexadecimal, ASCII) con filas de 16 bytes.
- **RF18 - Coloreado dinámico:** El sistema debe aplicar códigos de color diferenciados para: bytes en estado de fábrica (blanco), bytes modificados (azul), bytes protegidos (rojo).
- **RF19 - Navegación por páginas:** Para tarjetas SLE5528, el sistema debe proporcionar navegación entre las 4 páginas de memoria mediante botones dedicados (P0-P3).
- **RF20 - Actualización en tiempo real:** La vista de memoria debe actualizarse automáticamente tras cada operación APDU que modifique el contenido.

Integración con hardware físico

- **RF21 - Lectura desde tarjeta física:** El sistema debe permitir leer tarjetas físicas mediante lectores PC/SC compatibles, creando una tarjeta virtual con el contenido leído dependiendo del tipo de tarjeta.
- **RF22 - Escritura a tarjeta física:** El sistema debe permitir transferir el contenido de una tarjeta virtual a una tarjeta física, validando compatibilidad de tipos y ejecutando verificación post-escritura.
- **RF23 - PSC personalizado en lectura:** El sistema debe soportar PSCs personalizados para acceder a zonas protegidas de tarjetas físicas, conservando este PSC en la tarjeta virtual creada.
- **RF24 - Protección en escritura física:** El sistema debe omitir las dos primeras filas de memoria (0x00-0x1F) al escribir en tarjetas físicas para evitar sobrescribir zonas críticas.
- **RF25 - Detección de lectores:** El sistema debe detectar automáticamente lectores PC/SC conectados y mostrar mensajes de error claros cuando no estén disponibles.

Sistema de logging

- **RF26 - Registro cronológico:** El sistema debe mantener un log completo de todas las operaciones ejecutadas con timestamps precisos.
- **RF27 - Categorización por colores:** Los mensajes del log deben categorizarse mediante colores: negro (info), azul/morado (APDUs y ASCII), verde (success y hexadecimal), amarillo (warnings), rojo (errores).
- **RF28 - Exportación de log:** El sistema debe permitir exportar el contenido completo del log a archivos de texto para documentación.
- **RF29 - Limpieza de log:** El sistema debe permitir limpiar el log con confirmación del usuario, sin afectar el estado de las tarjetas.

Funcionalidades avanzadas

- **RF30 - Configuración de usuario:** El sistema debe permitir configurar información de identificación (nombre, descripción) que se incluya en tarjetas guardadas y logs exportados.
- **RF31 - Modo administrador:** El sistema debe incluir un modo administrativo protegido que habilite funcionalidades especiales (cambio de PSC en tarjetas físicas).
- **RF32 - Adaptación de interfaz:** El sistema debe ofrecer modo Small Screen Form Factor para pantallas de menor resolución, reorganizando elementos sin pérdida de funcionalidad.
- **RF33 - Referencia APDU integrada:** El sistema debe proporcionar documentación integrada de todos los comandos APDU disponibles con sus parámetros y formatos.
- **RF34 - Atajos de teclado:** El sistema debe implementar atajos de teclado para operaciones frecuentes (1-8 para comandos APDU, P+número para páginas, C+número para cambio de tarjeta).

4.2.2. Requisitos no funcionales

Los requisitos no funcionales establecen las cualidades y restricciones del sistema que no están directamente relacionadas con funcionalidades específicas, pero que son críticas para su éxito.

Usabilidad

- **RNF1 - Interfaz intuitiva:** La interfaz debe ser comprensible para estudiantes sin experiencia previa con tarjetas inteligentes, utilizando mensajes de ayuda y descripciones de los tipos de datos que se necesitan en todo momento.
- **RNF2 - Retroalimentación visual:** Todas las operaciones deben proporcionar retroalimentación visual inmediata mediante cambios de color, mensajes en el log y actualizaciones de la interfaz.
- **RNF3 - Prevención de errores:** El sistema debe prevenir operaciones peligrosas mediante validaciones, mensajes de advertencia y diálogos de confirmación para acciones irreversibles.
- **RNF4 - Consistencia visual:** La interfaz debe mantener un diseño coherente con iconografía clara, paleta de colores distintiva y organización lógica de paneles.
- **RNF5 - Curva de aprendizaje reducida:** Un estudiante nuevo debe poder realizar operaciones básicas (crear tarjeta, ejecutar comandos APDU, visualizar memoria) en menos de 5 minutos tras el primer contacto con la aplicación.

Fiabilidad

- **RNF6 - Manejo de errores:** El sistema debe capturar y manejar elegantemente todos los errores posibles (archivos corruptos, lectores desconectados, comandos inválidos) sin terminar abruptamente.

- **RNF7 - Validación de entradas:** Todas las entradas de usuario deben validarse antes de procesarse, mostrando mensajes de error específicos cuando sean inválidas.

Seguridad

- **RNF8 - Simulación fiel del hardware:** El comportamiento de las tarjetas virtuales debe replicar exactamente el de las tarjetas físicas, incluyendo bloqueos irreversibles por contador de errores agotado.
- **RNF9 - Protección de funciones administrativas:** Las funcionalidades de modo administrador (cambiar PSC físico) deben requerir confirmación explícita y advertir claramente que son solo para entorno educativo.
- **RNF10 - Validación de PSC:** La verificación de PSC debe implementarse byte a byte con mensajes claros sobre intentos restantes y consecuencias del bloqueo.

Conformidad

- **RNF11 - Estándar ISO 7816-4:** Los comandos APDU deben cumplir con la estructura y códigos de respuesta definidos en la norma ISO/IEC 7816-4 para tarjetas de memoria.
- **RNF12 - Compatibilidad PC/SC:** La comunicación con lectores debe seguir el estándar PC/SC (Personal Computer/Smart Card) para máxima interoperabilidad.
- **RNF13 - Formato de archivos legible:** Los archivos .txt generados deben ser legibles por humanos para facilitar inspección manual y depuración.

4.2.3. Priorización de requisitos

Los requisitos se han clasificado según su prioridad para el desarrollo:

| Prioridad | Requisitos |
|----------------|---|
| Crítica | RF1-RF13 (gestión de tarjetas y comandos APDU) RF14-RF16 (máquina de estados) RF17-RF20 (visualización) RNF1-RNF3 (usabilidad básica) RNF6-RNF7 (fiabilidad) RNF8 (simulación fiel del hardware) RNF11 (conformidad ISO 7816-4) |
| Alta | RF21-RF25 (integración hardware) RF26-RF29 (logging) RNF4-RNF5 (usabilidad avanzada) RNF9-RNF10 (seguridad) RNF12 (compatibilidad PC/SC) |
| Media | RF30-RF34 (funcionalidades avanzadas) RNF13 (formato de archivos legible) |

Tabla 4.1. Priorización de requisitos de CardSIM

Esta priorización permitió enfocar el desarrollo inicial en las funcionalidades core (tarjetas virtuales y comandos APDU) antes de añadir características avanzadas como la integración con hardware físico o el modo administrador.

4.3. Planificación temporal

4.3.1. Cronograma de desarrollo

El desarrollo del proyecto siguió una secuencia estructurada en paquetes de trabajo bien definidos, aunque con cierto solapamiento temporal entre fases para optimizar el tiempo de desarrollo:

1. Investigación y análisis de requisitos (Junio, semanas 1-2)

Realización de un estudio detallado de las tarjetas SLE5542 y SLE5528, análisis del estándar ISO 7816-4, investigación de las limitaciones de las herramientas existentes (c3tool y CardSim antiguo), y comunicación con

el tutor para identificar las necesidades educativas reales de la asignatura. Se documentaron casos de bloqueos de tarjetas en cursos anteriores y se establecieron los objetivos del proyecto.

2. Especificación de requisitos y planificación (Junio, semanas 3-4)

Definición formal de requisitos funcionales y no funcionales, priorización de características, selección de tecnologías (Python, Tkinter, pycard), y establecimiento del cronograma detallado del proyecto. Se definieron los objetivos y las métricas de evaluación.

3. Diseño de arquitectura del sistema (Julio, semana 1)

Diseño de la arquitectura modular del sistema con separación clara entre capas: Core (lógica de negocio), GUI (interfaz gráfica) y Utils (utilidades). Definición de interfaces entre módulos, diseño de la máquina de estados de las tarjetas, y especificación del modelo de datos para tarjetas virtuales.

4. Desarrollo de funcionalidades Core (Julio, semanas 2-5)

Implementación de las clases fundamentales: CardSession, MemoryManager, APDUHandler y SessionManager. Desarrollo de la lógica de tarjetas virtuales, implementación de los 8 comandos APDU según el estándar ISO 7816-4, y creación del sistema de guardado/carga de tarjetas en formato .txt. Durante esta fase se solapó parcialmente con el inicio del desarrollo de la GUI.

5. Desarrollo de interfaz gráfica (Julio-Agosto, semanas 3-7)

Implementación de la ventana principal con Tkinter, desarrollo de paneles especializados (visualización de memoria, log de comandos, panel APDU), implementación del sistema de coloreado dinámico, y creación de diálogos para operaciones de usuario. Esta fase se desarrolló en paralelo con las últimas semanas de desarrollo Core.

6. Integración del sistema de logging (Agosto, semana 1)

Desarrollo del sistema completo de registro de operaciones con categorización por colores, timestamps, y funcionalidades de exportación y limpieza del log.

7. Gestión de múltiples tarjetas (Agosto, semana 2)

Implementación del SessionManager para soporte de hasta 8 tarjetas simultáneas, desarrollo del panel CardExplorer, y sistema de navegación entre tarjetas con atajos de teclado.

8. Integración con hardware físico (Septiembre, semanas 1-4)

Implementación del PhysicalCardHandler utilizando la librería pycard, desarrollo de funcionalidades de lectura desde tarjetas físicas, implementación de escritura a tarjetas físicas con verificación automática, y manejo de errores de comunicación con lectores PC/SC. Se realizaron pruebas exhaustivas con el lector ACR39U.

9. Refinamiento de experiencia de usuario (Octubre, semanas 1-3)

Optimización de la interfaz gráfica: ajustes de colores, espaciado y distribución de elementos. Implementación del modo Small Screen Form Factor para adaptación a diferentes resoluciones. Desarrollo del panel de Actions con funcionalidades complementarias (exportación de logs, referencia APDU, configuración de usuario). Implementación del modo administrador para funciones avanzadas.

10. Validación en entorno real (Octubre, semanas 3-4)

Pruebas con estudiantes de la asignatura en condiciones reales de laboratorio. Recopilación de feedback sobre usabilidad, detección de errores y puntos de mejora. Observación del comportamiento de usuarios sin experiencia previa con la herramienta.

11. Correcciones y ajustes finales (Octubre, semana 5)

Implementación de mejoras basadas en el feedback recibido, corrección de errores detectados durante las pruebas, refinamiento de mensajes de validación y advertencias, y optimización del rendimiento. Generación del ejecutable final con PyInstaller.

12. Documentación del proyecto (Octubre semana 4 - Noviembre semana 1)

Redacción de la memoria del proyecto: introducción, objetivos, marco teórico, implementación, resultados y conclusiones. Preparación de figuras, capturas de pantalla y diagramas. Revisión final de la documentación. Esta fase se inició en paralelo con la validación en aula y se extendió hasta principios de noviembre.

13. Preparación de la defensa (Noviembre, semana 2)

Preparación de materiales para la presentación y defensa del proyecto: elaboración de diapositivas, ensayo de la presentación, y preparación de demostración en vivo de la aplicación.

La Figura 4.1 muestra gráficamente la distribución temporal de estas fases, evidenciando los períodos de solapamiento entre tareas y las dependencias entre etapas del desarrollo.

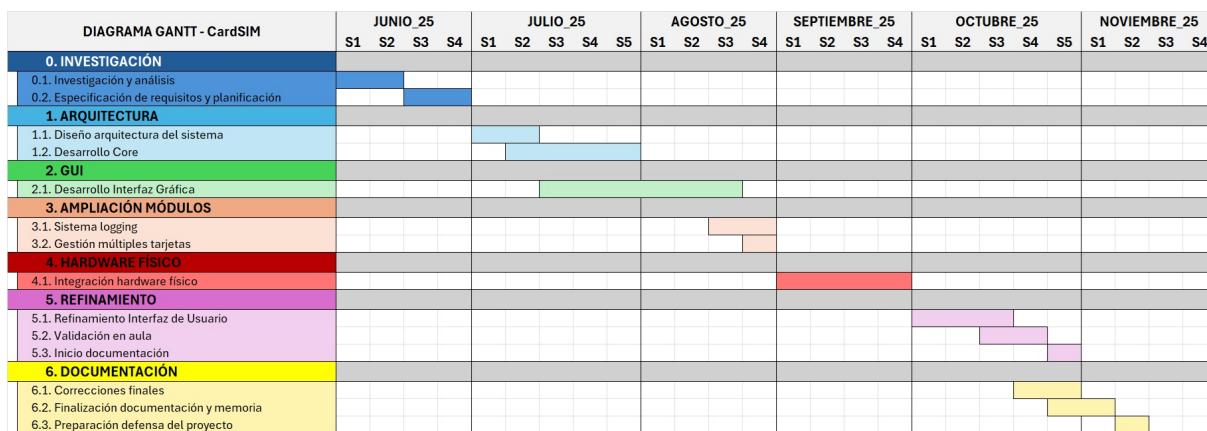


Figura 4.1. Diagrama de Gantt del proyecto

4.3.2. Hitos principales

Los hitos del proyecto marcan los puntos de control que permitieron validar el progreso y asegurar la calidad del desarrollo:

1. **Hito 1 (fin de junio):** Especificación completa de requisitos y diseño de la arquitectura
 2. **Hito 2 (fin de julio):** Prototipo funcional con interfaz gráfica y comandos APDU básicos operativos
 3. **Hito 3 (mediados de agosto):** Funcionalidades core completadas (tarjetas virtuales, guardado/carga, gestión múltiple)
 4. **Hito 4 (fin de septiembre):** Integración completa con hardware físico y comunicación con lectores PC/SC

5. **Hito 5 (fin de octubre)**: Sistema completo validado en pruebas de aula con estudiantes reales
6. **Hito 6 (mediados de noviembre)**: Documentación completa y proyecto finalizado para presentación

5.

Marco teórico

Este capítulo presenta los fundamentos teóricos necesarios para comprender el funcionamiento de las tarjetas inteligentes SLE5528 y SLE5542, así como el protocolo de comunicación APDU utilizado para interactuar con ellas.

5.1. Tarjetas inteligentes SLE5528 y SLE5542

Las tarjetas SLE5528 y SLE5542 son dispositivos de memoria EEPROM fabricados por Infineon (anteriormente Siemens) que incorporan mecanismos de protección mediante un Código de Seguridad Programable (PSC, del inglés *Programmable Security Code*), también conocido como PIN. Estos dispositivos permiten el almacenamiento seguro de datos con diferentes niveles de protección contra escritura.

Ambos modelos presentan características comunes pero difieren en su capacidad de almacenamiento y en la configuración de sus mecanismos de seguridad, lo que las hace adecuadas para diferentes aplicaciones prácticas y educativas.

5.2. Protocolo APDU (ISO 7816-4)

5.2.1. Fundamentos del protocolo

El protocolo APDU (*Application Protocol Data Unit*) está definido en la norma ISO 7816-4. Se trata de un protocolo de alto nivel correspondiente a la capa de aplicación del modelo OSI, que define el formato de los mensajes intercambiados entre un host (lector) y una SmartCard durante su comunicación.

La norma establece dos formatos de mensaje principales:

- **C-APDU (Command APDU)**: Mensajes utilizados por el host para enviar órdenes a la tarjeta.
- **R-APDU (Response APDU)**: Mensajes utilizados por la tarjeta para enviar respuestas al host.

El protocolo establece una relación maestro-esclavo donde la tarjeta nunca inicia la comunicación de forma autónoma, sino que responde únicamente a los comandos enviados previamente por el host. El diálogo tiene lugar en modo half-duplex (comunicación bilateral alternada).

5.2.2. Secuencia de reset y ATR

La primera acción que se debe realizar sobre la tarjeta es una operación de reset, que consiste en aplicar las señales eléctricas apropiadas según el estándar ISO 7816-3. Esta operación la realiza automáticamente el lector al insertar la tarjeta, siguiendo una secuencia específica de activación.

Proceso de activación de la tarjeta (ISO 7816-3)

Al insertar la tarjeta en el lector, se ejecuta automáticamente la siguiente secuencia de activación:

1. El lector (host) aplica alimentación entre los pines Vcc y GND. La tensión puede ser de 1.8V, 3.3V o 5V dependiendo del tipo de tarjeta.
2. El lector pone la entrada RST (Reset) a nivel bajo, realizando un POR (Power On Reset).
3. El lector aplica señal de reloj por el pin Ck para sincronizar la comunicación.
4. El lector se mantiene unos milisegundos en espera de que la tarjeta ponga el pin I/O a nivel alto, indicando que está lista para enviar su ATR (Answer To Reset).

Si la tarjeta no responde poniendo a nivel alto su pin I/O, el lector incrementa la tensión de alimentación al siguiente valor estándar ($1.8V \rightarrow 3.3V \rightarrow 5V$) y repite toda la secuencia anterior hasta obtener respuesta o agotar las opciones de tensión.

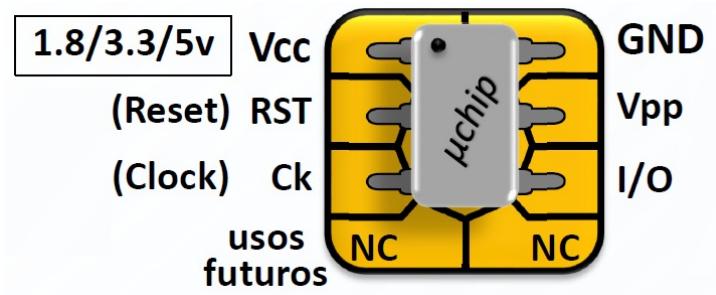


Figura 5.1. Microprocesador de una tarjeta de contacto

La Figura 5.1 muestra el microprocesador de una tarjeta inteligente con sus contactos físicos visibles, incluyendo los pines Vcc, GND, RST, Ck e I/O que participan en el proceso de activación.

Contenido del ATR

Una vez establecida la comunicación, la tarjeta responde con su ATR (*Answer To Reset* o "histórico"), que contiene información fundamental sobre sus características y capacidades:

- Identificación del fabricante del chip.
- Protocolo de transmisión que soporta ($T=0$ para transmisión orientada a bytes, $T=1$ para transmisión orientada a bloques).
- Convención directa o inversa (nivel lógico “1” alto o bajo).
- Configuración de bits de inicio y parada.
- Frecuencia de reloj de transmisión.
- Configuración de paridad.

Tras el ATR puede producirse, o no, una negociación de parámetros (PPS, *Protocol and Parameters Selection*), dependiendo de si el lector acepta la configuración propuesta por la tarjeta o solicita modificaciones.

5.2.3. Estructura de los mensajes APDU

Los mensajes C-APDU y R-APDU presentan la siguiente estructura:

Estructura del C-APDU (Command)

| Campo | Descripción |
|-------|--|
| CLA | Clase de instrucción, identifica la entidad dentro de la tarjeta |
| INS | Código de operación de la instrucción a ejecutar |
| P1 | Primer parámetro de la instrucción |
| P2 | Segundo parámetro de la instrucción |
| Lc | Longitud del campo de datos enviados (en bytes) |
| DATA | Datos enviados a la tarjeta |
| Le | Longitud esperada de datos en la respuesta (en bytes) |

Tabla 5.1. Estructura del mensaje C-APDU

Estructura del R-APDU (Response)

| Campo | Descripción |
|-------|-----------------------------------|
| DATA | Datos devueltos por la tarjeta |
| SW1 | Primer byte del código de estado |
| SW2 | Segundo byte del código de estado |

Tabla 5.2. Estructura del mensaje R-APDU

El campo SW1-SW2 es un código de dos bytes que informa sobre el resultado de la ejecución. El valor `90 00h` indica éxito, mientras que otros valores identifican diferentes tipos de errores.

5.2.4. Casos de uso de APDU

En la práctica, dependiendo de las capacidades del chip integrado en la tarjeta, se pueden presentar cuatro situaciones posibles en la comunicación AP-

DU:

1. **Caso 1:** Comando sin datos de entrada ni de salida
2. **Caso 2:** Comando sin datos de entrada, con datos de salida
3. **Caso 3:** Comando con datos de entrada, sin datos de salida
4. **Caso 4:** Comando con datos de entrada y de salida

5.3. Características de la tarjeta SLE5542

La tarjeta SLE5542 presenta las siguientes características técnicas:

5.3.1. Capacidad y organización de memoria

- Memoria EEPROM de 256 bytes (direcciones 00h a FFh)
- Memoria organizada en bytes de 8 bits
- Direcciones 00h a 1Fh: zona de memoria protegida (32 bytes)
- Direcciones 20h a FFh: zona de memoria de usuario (224 bytes)

5.3.2. Mecanismo de protección

Las 32 direcciones más bajas (00h a 1Fh) pueden protegerse contra escritura de forma individual e irreversible mediante un registro de 32 bits denominado *Write Protection Bits*:

- Bit = 1: Escritura habilitada (*WRITE_ON*)
- Bit = 0: Escritura deshabilitada de forma permanente (*WRITE_OFF*)

Algunas direcciones vienen protegidas de fábrica, especialmente aquellas que contienen el *Application ID* y el *Chip Code* del fabricante.

5.3.3. Código de seguridad (PSC)

- **Tamaño del PSC:** 3 bytes
- **Valor de fábrica:** FF FF FF
- **Intentos disponibles:** 3 intentos
- **Contador de errores:** Se inicializa a 3 tras una presentación correcta

El contador de errores se decrementa con cada presentación incorrecta del PSC. Cuando alcanza el valor 0, la escritura en la tarjeta queda bloqueada de forma irreversible, permitiendo únicamente operaciones de lectura.

5.4. Características de la tarjeta SLE5528

La tarjeta SLE5528 es un modelo de mayor capacidad con las siguientes características:

5.4.1. Capacidad y organización de memoria

- Memoria EEPROM de 1024 bytes (1 KB)
- Organización en páginas y direcciones
- Mayor espacio disponible para almacenamiento de datos y prácticas educativas

5.4.2. Mecanismo de protección

A diferencia de la SLE5542, cada dirección de memoria de la SLE5528 dispone de un bit de protección individual contra escritura:

- Bit de protección = 1: Escritura habilitada (*WRITE_ON*)

- Bit de protección = 0: Escritura deshabilitada permanentemente (*WRI-TE_OFF*)

5.4.3. Código de seguridad (PSC)

- **Tamaño del PSC:** 2 bytes
- **Valor de fábrica:** FF FF
- **Intentos disponibles:** 8 intentos
- **Almacenamiento del PSC:** Direcciones 3FEh y 3FFh

El mayor número de intentos (8 frente a 3) proporciona una mayor flexibilidad durante el proceso de aprendizaje, reduciendo el riesgo de bloqueo permanente.

5.5. Comandos APDU para tarjetas de memoria

A continuación se describen los comandos APDU específicos para las tarjetas SLE5528 y SLE5542, indicando las particularidades de cada modelo.

5.5.1. SELECT_CARD_TYPE

Este comando es obligatorio y permite activar y seleccionar el modelo de tarjeta. Debe ser siempre el primer comando enviado tras el reset.

SLE5542

| CLA | INS | P1 | P2 | Lc | Data |
|-----|-----|-----|-----|-----|------|
| FFh | A4h | 00h | 00h | 01h | 06h |

Tabla 5.3. Comando SELECT_CARD_TYPE para SLE5542

SLE5528

| CLA | INS | P1 | P2 | Lc | Data |
|-----|-----|-----|-----|-----|------|
| FFh | A4h | 00h | 00h | 01h | 05h |

Tabla 5.4. Comando SELECT_CARD_TYPE para SLE5528

Respuesta esperada:

| SW1 | SW2 |
|-----|-----|
| 90h | 00h |

Tabla 5.5. Respuesta exitosa SELECT_CARD_TYPE

5.5.2. READ_MEMORY_CARD

Este comando permite leer un bloque de memoria de Le bytes a partir de una dirección específica.

SLE5542

| CLA | INS | P1 | P2 | Le |
|-----|-----|-----|----------|----------|
| FFh | B0h | 00h | @Address | Longitud |

Tabla 5.6. Comando READ_MEMORY_CARD para SLE5542

Donde @Address indica la dirección de inicio de lectura ($00h \leq @ \leq FFh$).

SLE5528

| CLA | INS | P1 | P2 | Le |
|-----|-----|----------|----------|----------|
| FFh | B0h | MSB_Addr | LSB_Addr | Longitud |

Tabla 5.7. Comando READ_MEMORY_CARD para SLE5528

Para la SLE5528, la dirección se especifica mediante dos bytes:

- MSB_Addr = $0000\ 00A_9A_8b$
- LSB_Addr = $A_7A_6A_5A_4\ A_3A_2A_1A_0b$

Respuesta esperada (SLE5542):

| BYTE 1 | ... | BYTE N | PROT 1-4 | SW1 | SW2 |
|--------|-----|--------|-----------------|-----|-----|
| Datos | ... | Datos | Protection Bits | 90h | 00h |

Tabla 5.8. Respuesta READ_MEMORY_CARD SLE5542

Respuesta esperada (SLE5528):

| BYTE 1 | ... | BYTE N | SW1 | SW2 |
|--------|-----|--------|-----|-----|
| Datos | ... | Datos | 90h | 00h |

Tabla 5.9. Respuesta READ_MEMORY_CARD SLE5528

5.5.3. PRESENT_CODE_MEMORY_CARD

Este comando se utiliza para presentar el PSC (PIN) a la tarjeta, habilitando las operaciones de escritura. Solo es necesario presentarlo una vez por sesión (mientras la tarjeta permanezca en el lector).

SLE5542

| CLA | INS | P1 | P2 | Lc | PSC (3 bytes) | | |
|-----|-----|-----|-----|-----|---------------|-------|-------|
| FFh | 20h | 00h | 00h | 03h | Byte1 | Byte2 | Byte3 |

Tabla 5.10. Comando PRESENT_CODE para SLE5542

SLE5528

| CLA | INS | P1 | P2 | Lc | PSC (2 bytes) | |
|-----|-----|-----|-----|-----|---------------|-------|
| FFh | 20h | 00h | 00h | 02h | Byte1 | Byte2 |

Tabla 5.11. Comando PRESENT_CODE para SLE5528

Respuesta esperada (SLE542):

| SW1 | SW2 (ErrorCnt) |
|-----|--|
| 90h | 07h = Correcto |
| 90h | 03h, 01h = Incorrecto (2 ó 1 intentos restantes) |
| 90h | 00h = Tarjeta bloqueada |

Tabla 5.12. Respuesta PRESENT_CODE SLE542

Respuesta esperada (SLE5528):

| SW1 | SW2 (ErrorCnt) |
|-----|--|
| 90h | FFh = Correcto |
| 90h | 7Fh, 7Eh, 7Ch, 78h, 70h, 60h, 40h = Intentos restantes |
| 90h | 00h = Tarjeta bloqueada |

Tabla 5.13. Respuesta PRESENT_CODE SLE5528

Para la SLE5528, el contador de errores utiliza 8 bits, cada uno representando un intento disponible. Por ejemplo:

- FFh = 1111 1111b = 8 intentos (verificación correcta)
- 7Eh = 0111 1110b = 6 intentos restantes
- 00h = 0000 0000b = 0 intentos (bloqueada)

5.5.4. WRITE_MEMORY_CARD

Este comando permite escribir Lc bytes en la memoria a partir de una dirección específica. Requiere presentación previa y exitosa del PSC.

SLE5542

| CLA | INS | P1 | P2 | Lc | Data |
|------------|------------|-----------|-----------|-----------|------------------|
| FFh | D0h | 00h | @Address | Longitud | Bytes a escribir |

Tabla 5.14. Comando WRITE_MEMORY_CARD para SLE5542**SLE5528**

| CLA | INS | P1 | P2 | Lc | Data |
|------------|------------|-----------|-----------|-----------|------------------|
| FFh | D0h | MSB_Addr | LSB_Addr | Longitud | Bytes a escribir |

Tabla 5.15. Comando WRITE_MEMORY_CARD para SLE5528

Respuesta esperada (ambas tarjetas):

| SW1 | SW2 |
|------------|------------|
| 90h | 00h |

Tabla 5.16. Respuesta WRITE_MEMORY_CARD**5.5.5. READ_ERROR_COUNTER_MEMORY_CARD**

Este comando permite leer el contador de intentos fallidos al presentar el PSC.

SLE5542

En la SLE5542, el contador de errores indica cuántos intentos de presentación del PSC quedan disponibles. Cuando llega a cero, la tarjeta queda bloqueada permanentemente.

| CLA | INS | P1 | P2 | Le |
|------------|------------|-----------|-----------|-----------|
| FFh | B1h | 00h | 00h | 04h |

Tabla 5.17. Comando READ_ERROR_COUNTER para SLE5542

Respuesta SLE5542:

| ERRCNT | DUMMY1 | DUMMY2 | DUMMY3 | SW1 | SW2 |
|---------------|---------------|---------------|---------------|------------|------------|
| ErrorCounter | xx | xx | xx | 90h | 00h |

Tabla 5.18. Respuesta READ_ERROR_COUNTER SLE5542

El campo ERRCNT muestra el número de intentos restantes (máximo 3).

SLE5528

En la SLE5528, el contador de errores se representa en 8 bits, donde cada presentación incorrecta del PSC desactiva un bit (0). Cuando todos los bits están a cero, la tarjeta queda bloqueada.

| CLA | INS | P1 | P2 | Le |
|------------|------------|-----------|-----------|-----------|
| FFh | B1h | 00h | 00h | 03h |

Tabla 5.19. Comando READ_ERROR_COUNTER para SLE5528**Respuesta SLE5528:**

| ERRCNT | DUMMY1 | DUMMY2 | SW1 | SW2 |
|---------------|---------------|---------------|------------|------------|
| ErrorCounter | xx | xx | 90h | 00h |

Tabla 5.20. Respuesta READ_ERROR_COUNTER SLE5528

El campo ERRCNT contiene 8 bits que representan los intentos disponibles. Por ejemplo, FFh indica 8 intentos, FEh indica 7 intentos restantes, etc.

5.5.6. READ_PROTECTION_BITS

Este comando permite leer el contenido del registro de bits de protección contra escritura.

SLE5542

Para la SLE5542, este comando lee los 32 bits de protección de las primeras 32 direcciones.

| CLA | INS | P1 | P2 | Le |
|-----|-----|-----|-----|-----|
| FFh | B2h | 00h | 00h | 04h |

Tabla 5.21. Comando READ_PROTECTION_BITS para SLE5542

Respuesta SLE5542:

| PROT1 | PROT2 | PROT3 | PROT4 | SW1 | SW2 |
|----------|-----------|------------|------------|-----|-----|
| Bits 1-8 | Bits 9-16 | Bits 17-24 | Bits 25-32 | 90h | 00h |

Tabla 5.22. Respuesta READ_PROTECTION_BITS SLE5542

Los bits de protección se organizan como: P8 P7 P6 P5 P4 P3 P2 P1 en PROT1, P16 P15 P14 P13 P12 P11 P10 P9 en PROT2, y así sucesivamente.

SLE5528

Para la SLE5528, este comando puede leer cualquier conjunto de bits de protección especificando la dirección inicial y la longitud.

| CLA | INS | P1 | P2 | Le |
|-----|-----|----------|----------|-------|
| FFh | B2h | MSB_Addr | LSB_Addr | MEM_L |

Tabla 5.23. Comando READ_PROTECTION_BITS para SLE5528

Donde MEM_L se calcula como: $MEM_L = 1 + INT[(número_de_bits - 1)/8]$

Respuesta SLE5528:

| BYTE 1 | ... | BYTE N | SW1 | SW2 |
|-----------------|-----|-----------------|-----|-----|
| Protection Bits | ... | Protection Bits | 90h | 00h |

Tabla 5.24. Respuesta READ_PROTECTION_BITS SLE5528

Los bits de protección se devuelven en bytes, donde cada bit representa el estado de protección de una dirección específica (1 = no protegida, 0 = protegida).

5.5.7. WRITE_PROTECT_MEMORY_CARD

Este comando permite proteger de forma permanente e irreversible direcciones de memoria específicas. El proceso compara los bytes especificados con el contenido actual de la memoria, y si coinciden, desactiva la escritura en esas posiciones.

SLE5542

| CLA | INS | P1 | P2 | Lc | Data |
|------------|------------|-----------|-----------|-----------|------------------|
| FFh | D1h | 00h | @Address | Longitud | Bytes a comparar |

Tabla 5.25. Comando WRITE_PROTECT para SLE5542 (solo @00h-1Fh)

Importante: En la SLE5542, solo se pueden proteger las direcciones 00h a 1Fh.

SLE5528

| CLA | INS | P1 | P2 | Lc | Data |
|------------|------------|-----------|-----------|-----------|------------------|
| FFh | D1h | MSB_Addr | LSB_Addr | Longitud | Bytes a comparar |

Tabla 5.26. Comando WRITE_PROTECT para SLE5528

Importante: En la SLE5528, se pueden proteger todas las direcciones de memoria.

Respuesta esperada (ambas tarjetas):

| SW1 | SW2 |
|-----|-----|
| 90h | 00h |

Tabla 5.27. Respuesta WRITE_PROTECT

5.5.8. CHANGE_PSC_MEMORY_CARD

Este comando permite cambiar el PSC de la tarjeta. Es fundamental haber presentado correctamente el PSC actual antes de ejecutar este comando.

SLE5542

| CLA | INS | P1 | P2 | Lc | Nuevo PSC | | |
|-----|-----|-----|-----|-----|-----------|-------|-------|
| FFh | D2h | 00h | 01h | 03h | Byte1 | Byte2 | Byte3 |

Tabla 5.28. Comando CHANGE_CODE para SLE5542

SLE5528

La SLE5528 no dispone de un comando específico para cambiar el PSC. En su lugar, se utiliza el comando WRITE_MEMORY_CARD para escribir directamente en las direcciones 3FEh y 3FFh donde se almacena el PSC.

| CLA | INS | P1 | P2 | Lc | Nuevo PSC | |
|-----|-----|-----|-----|-----|-----------|-------|
| FFh | D0h | 03h | FEh | 02h | Byte1 | Byte2 |

Tabla 5.29. Cambio de PSC en SLE5528 mediante WRITE_MEMORY

Advertencia: El cambio de PSC es irreversible. Si se olvida el nuevo código, la tarjeta quedará bloqueada permanentemente para operaciones de escritura.

Respuesta esperada (ambas tarjetas):

| SW1 | SW2 |
|-----|-----|
| 90h | 00h |

Tabla 5.30. Respuesta CHANGE_CODE

5.5.9. Ejemplo práctico de operación

A continuación se presenta un ejemplo de secuencia de comandos para escribir información de usuario en una tarjeta:

Objetivo: Escribir los campos NAME1, NAME2 y NAME3 en las direcciones 20h, 30h y 40h respectivamente.

Secuencia de comandos (SLE5542):

1. FF A4 00 00 01 06 - Seleccionar tarjeta
2. FF 20 00 00 03 FF FF FF - Presentar PSC de fábrica
3. FF D0 00 20 10 4E 41 4D 45 31 3A 4A 4F 53 45 ... - Escribir NAME1
4. FF D0 00 30 10 4E 41 4D 45 32 3A 47 55 54 49 ... - Escribir NAME2
5. FF D0 00 40 10 4E 41 4D 45 33 3A 46 45 52 4E ... - Escribir NAME3
6. FF B0 00 20 30 - Leer y verificar los datos escritos

Donde los datos hexadecimales corresponden a caracteres ASCII: 4E 41 4D 45 31 3A 4A 4F 53 45 = "NAME1:JOSE"

6.

Sistema desarrollado

Este capítulo describe el sistema CardSIM desarrollado como aplicación de escritorio para la simulación y gestión de tarjetas inteligentes SLE5528 y SLE5542. Se presenta la arquitectura del software, la organización modular del código, las tecnologías utilizadas y una descripción detallada de la interfaz gráfica y sus funcionalidades.

6.1. Arquitectura del sistema

6.1.1. Visión general

CardSIM ha sido desarrollado siguiendo una arquitectura modular basada en el patrón MVC (Modelo-Vista-Controlador) adaptado para aplicaciones de escritorio. La aplicación se estructura en tres capas principales:

- **Capa de presentación (GUI):** Interfaz gráfica de usuario basada en Tkinter
- **Capa de lógica interna (Core):** Gestión de sesiones, memoria y comunicación APDU
- **Capa de utilidades (Utils):** Funciones auxiliares, gestión de recursos y configuración

6.1.2. Tecnologías utilizadas

La aplicación ha sido desarrollada utilizando las siguientes tecnologías:

| Componente | Tecnología | Descripción |
|-------------|--------------|--|
| Lenguaje | Python 3.11 | Lenguaje principal de desarrollo |
| GUI | Tkinter | Biblioteca estándar para interfaces gráficas |
| Imágenes | Pillow (PIL) | Procesamiento y gestión de iconos PNG |
| Hardware | pyscard | Comunicación con lectores PC/SC (opcional) |
| Empaquetado | PyInstaller | Generación de ejecutables independientes |

Tabla 6.1. Tecnologías utilizadas en CardSIM

6.1.3. Estructura del proyecto

El proyecto está organizado en una estructura de directorios clara y modular:

6.2. Módulos principales

CardSIM está organizado en tres módulos principales que separan responsabilidades siguiendo principios de diseño de software:

- **Módulo Core:** Encapsula toda la lógica de la aplicación y simulación de tarjetas inteligentes. Incluye las clases SessionManager (gestión de sesiones múltiples), CardSession (estado individual de cada tarjeta), MemoryManager (abstracción de memoria EEPROM), APDUHandler (procesamiento del protocolo ISO 7816-4), y PhysicalCardHandler (integración con hardware real vía PC/SC).
- **Módulo GUI:** Implementa la interfaz gráfica completa usando Tkinter. La clase CardSimInterface coordina todos los componentes visuales, mientras que clases especializadas como NewCardDialog, ReadMemoryDialog, ProtectionBitsDialog, etc., gestionan ventanas emergentes para operaciones específicas. El componente CardExplorer proporciona visualización avanzada de memoria con navegación y búsqueda.
- **Módulo Utils:** Concentra utilidades transversales: Constants (configuración centralizada de tamaños, códigos APDU, colores), AppStates (máquinas de estados que controlan el flujo de la aplicación), ResourceManager (localización de recursos en scripts Python y ejecutables empaquetados), y UserConfig (gestión de identificación de usuario con persistencia portable).

Esta arquitectura de tres capas mantiene bajo acoplamiento entre componentes: la lógica de simulación es completamente independiente de la interfaz gráfica, permitiendo pruebas unitarias del Core sin inicializar Tkinter, y facilitando potenciales interfaces alternativas (CLI, web) que reutilizarían el mismo Core sin modificaciones.

Los detalles técnicos de implementación de cada clase se presentan integrados en la sección siguiente junto con las capturas de pantalla de la interfaz, facilitando la comprensión de cómo el código se materializa visualmente en la aplicación.

6.3. Interfaz de usuario integrada

La interfaz gráfica de CardSIM, implementada mediante la clase CardSimInterface, organiza todos los componentes visuales en una ventana principal estructurada en paneles especializados. La construcción de esta interfaz en el método `setup_ui()` divide la ventana en regiones funcionales bien delimitadas que permiten acceso rápido a todas las funcionalidades del simulador.

6.3.1. Ventana principal y distribución de paneles

Vista inicial - Distribución estándar

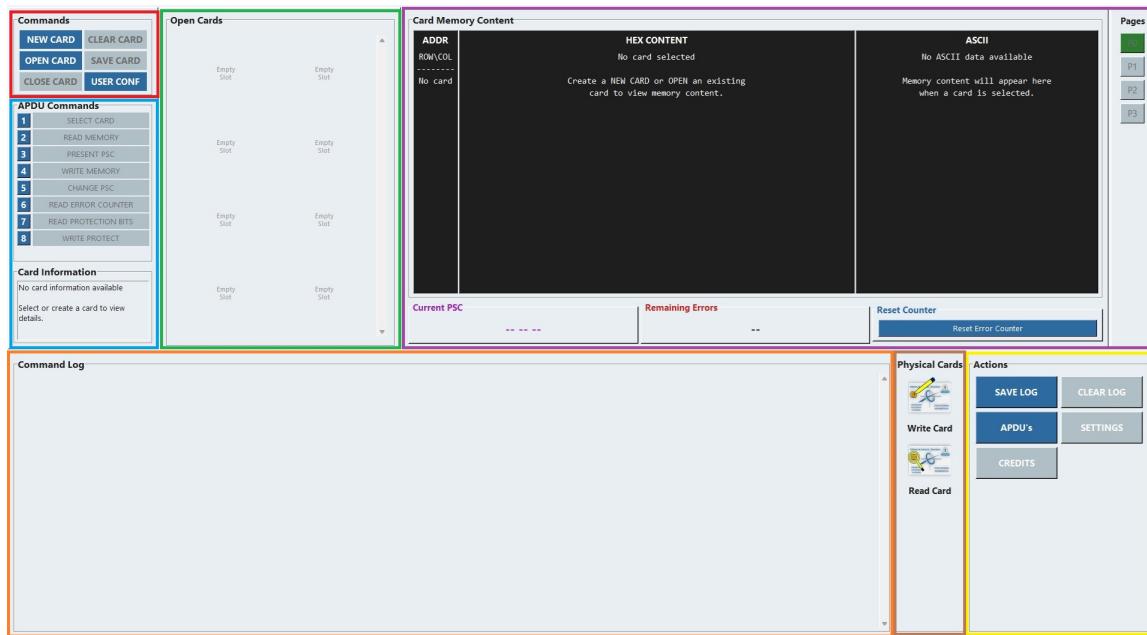


Figura 6.1. Vista inicial de CardSIM mostrando la distribución completa de paneles

La Figura 6.1 muestra la distribución completa de la interfaz con paneles recuadrados por colores para facilitar su identificación:

Zona superior izquierda:

- **Panel Commands** (recuadro rojo): Contiene seis botones para gestión

de tarjetas virtuales: New Card, Open Card, Save Card, Clear Card, Close Card y User Conf.

- **Panel APDU Commands** (recuadro azul, debajo de Commands): Presenta los 8 comandos del protocolo ISO 7816-4 organizados verticalmente con números del 1 al 8 que corresponden a atajos de teclado numéricos.
- **Panel Card Information** (recuadro azul, debajo de APDU Commands): Muestra metadatos de la tarjeta activa: nombre, tipo de tarjeta, estado de conexión (status), estado del PSC verificado y Error Counter actual.

Zona superior derecha:

- **Panel Open Cards** (recuadro verde, a la derecha de Commands/APDU Commands): Lista todas las tarjetas abiertas en la sesión con 8 slots disponibles. Permite cambiar entre tarjetas con un simple click sobre cualquiera de ellas.
- **Panel Card Memory Content** (recuadro morado, ocupando la mayor parte superior): Visualizador principal de memoria que muestra el contenido completo de la tarjeta activa. Incluye subpaneles auxiliares:
 - **Pages**: Botones para navegar entre las 4 páginas de memoria (solo disponible en SLE5528).
 - **Current PSC**: Muestra el código de seguridad actual de la tarjeta.
 - **Remaining Errors**: Indica los intentos restantes del contador de errores.
 - **Reset Counter**: Botón para reiniciar el contador de errores.

Zona inferior:

- **Panel Command Log** (recuadro naranja, izquierda inferior): Registro cronológico de todas las operaciones ejecutadas con marcas temporales y codificación por colores.
- **Panel Physical Cards** (recuadro marrón, centro inferior): Integración con hardware real mediante dos botones: Write Card (escritura a tarjeta física) y Read Card (lectura desde tarjeta física en el lector).

- **Panel Actions** (recuadro amarillo, derecha inferior): Funcionalidades adicionales organizadas en cuadrícula: Save Log, Clear Log, APDU's (teoría de Comercio Electrónico), Settings, Credits. El botón Change PSC permanece oculto hasta activar el modo administrador.

Vista con tarjetas cargadas

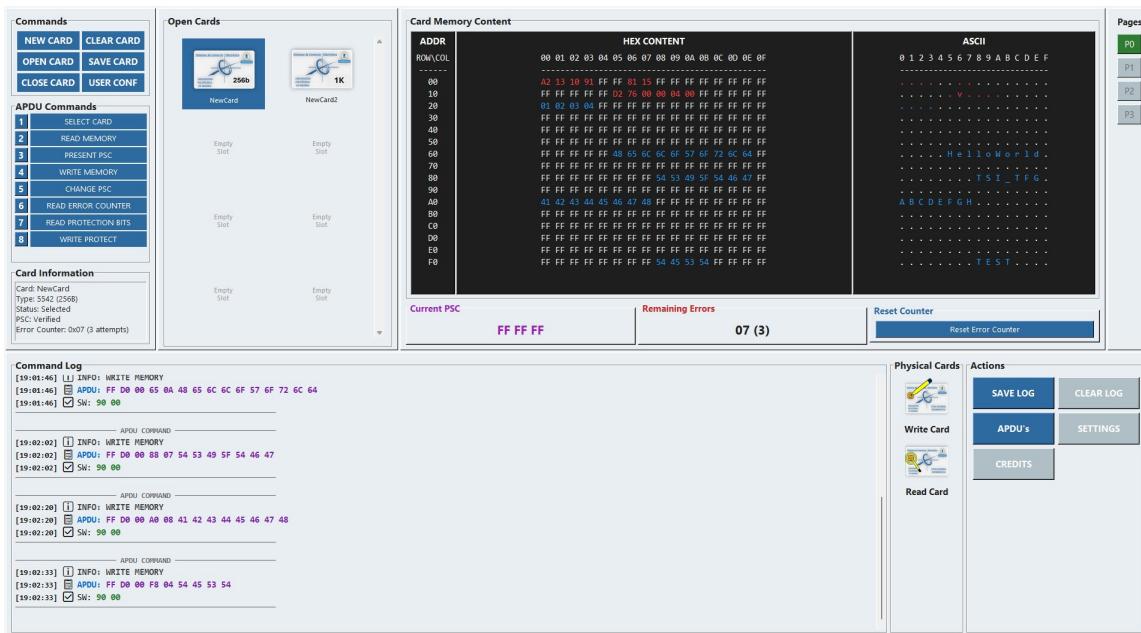


Figura 6.2. Vista de CardSIM con tarjetas cargadas y operaciones en curso

La Figura 6.2 muestra la aplicación con múltiples tarjetas cargadas y operaciones ejecutadas. Se observa el contenido de memoria poblado con datos hexadecimales y su representación ASCII, el panel Open Cards mostrando 2 tarjetas activas, y el Command Log con historial detallado de comandos APDU ejecutados y sus respuestas correspondientes.

Nota: Mientras que la Figura 6.1 (vista inicial) presenta una distribución limpia con recuadros de colores para facilitar la comprensión de la estructura organizativa de la interfaz, la Figura 6.2 contiene todos los datos y elementos mencionados en funcionamiento real, mostrando cómo se visualizan los paneles cuando la aplicación está en uso activo.

Coordinación de la interfaz y atajos de teclado: La clase CardSimInterface actúa como el controlador central de toda la interfaz gráfica, dirigiendo la in-

teracción entre componentes visuales y la lógica encapsulada en el módulo Core. Esta clase actúa como intermediario centralizado, donde todos los eventos de usuario (clicks de botones, entradas de texto, selecciones de menú) pasan por métodos de esta clase que coordinan las respuestas apropiadas.

El sistema de actualización funciona mediante notificaciones donde los cambios en el estado de la sesión activa provocan llamadas a métodos como `update_card_display()`, `update_button_states()`, y `update_cards_list()`. Estos métodos reconstruyen selectivamente las porciones afectadas de la interfaz sin redesplegar toda la ventana, optimizando el rendimiento y evitando parpadeos visuales molestos.

La interfaz implementa atajos de teclado para agilizar el trabajo: las teclas numéricas 1-8 ejecutan directamente los comandos APDU correspondientes cuando los botones están habilitados. Además, la combinación P+número permite cambiar entre páginas de memoria (P+0 a P+3 para las cuatro páginas de la SLE5528), mientras que C+número permite cambiar entre las tarjetas abiertas (C+1 a C+8). En las ventanas de diálogo, las teclas Enter y Escape facilitan la confirmación o cancelación rápida de operaciones.

El manejo de cierre de aplicación mediante `on_closing()` ejecuta una secuencia de limpieza ordenada: primero destruye todas las ventanas de diálogo abiertas para evitar referencias colgantes, luego cierra todas las sesiones activas liberando sus recursos asociados mediante SessionManager, elimina archivos temporales del sistema, y finalmente destruye la ventana raíz de Tkinter asegurando la terminación limpia del proceso.

6.3.2. Panel de comandos principales



Figura 6.3. Panel Commands con botones de gestión de archivos

El panel Commands (Figura 6.3) contiene seis botones fundamentales implementados por la clase CardSimInterface:

- **New Card:** invoca `new_card_dialog()` para crear una nueva tarjeta virtual
- **Open Card:** ejecuta `open_card_dialog()` para cargar tarjetas desde archivo
- **Save Card:** llama a `save_card_dialog()` para guardar la tarjeta activa
- **Clear Card:** reinicia el contenido de la tarjeta activa (memoria a 0xFF)
- **Close Card:** cierra y descarga la tarjeta activa de memoria
- **User Conf:** abre el diálogo de configuración de usuario

Estos métodos interactúan directamente con SessionManager para gestionar el ciclo de vida de las tarjetas virtuales.

Importante: CardSIM no realiza guardados automáticos. Todas las tarjetas creadas o cargadas permanecen en memoria (panel Open Cards) hasta ser cerradas explícitamente, pero solo se persisten a disco cuando el usuario ejecuta Save Card manualmente. Este diseño intencional evita sobrescrituras accidentales y da control total al usuario sobre qué cambios conservar.

Botón New Card - Crear nueva tarjeta

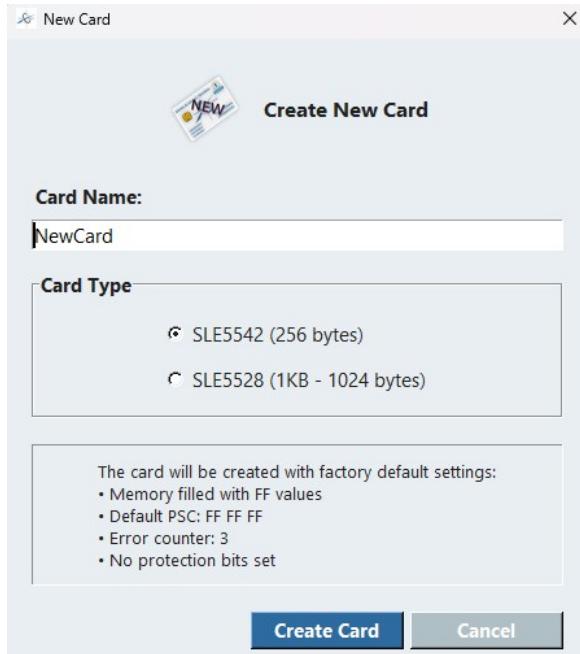


Figura 6.4. Diálogo creación SLE5542

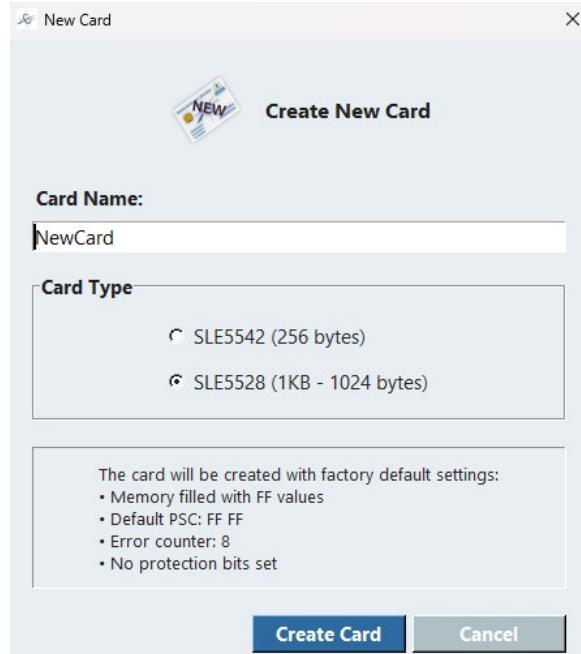


Figura 6.5. Diálogo creación SLE5528

La clase `NewCardDialog` presenta un formulario con botones de selección para el tipo de tarjeta y campo de entrada para el nombre. Al confirmar, el diálogo invoca el método `create_new_card_session()` de `SessionManager`, que instancia un `CardSession` nuevo y llama a `initialize_memory()` de `MemoryManager` para establecer la configuración de fábrica.

Para SLE5542 (Figura 6.4): 256 bytes inicializados en 0xFF, PSC de fábrica FF FF FF (3 bytes), contador de errores en 0x07 (3 intentos). Para SLE5528 (Figura 6.5): 1024 bytes organizados en 4 páginas, PSC de fábrica FF FF (2 bytes), contador en 0x7F (7 intentos).

Sistema modular de diálogos: CardSIM implementa un conjunto extenso de clases de diálogo especializadas (`NewCardDialog`, `ReadMemoryDialog`, `WriteMemoryDialog`, `PresentPSCDialog`, `ProtectionBitsDialog`, etc.), cada una encapsulando la interfaz y lógica de validación para una operación específica.

Este diseño modular del módulo GUI facilita mantenimiento y extensibilidad, ya que añadir nuevas funcionalidades implica crear una nueva clase de diálogo sin modificar código existente. La validación integrada previene nombres

vacíos o duplicados, mostrando mensajes de error inline sin cerrar el diálogo, permitiendo al usuario corregir inmediatamente. Al confirmar, el diálogo retorna un diccionario con los datos ingresados que la interfaz principal utiliza para invocar los métodos correspondientes de SessionManager.

Botón Open Card - Abrir tarjeta guardada

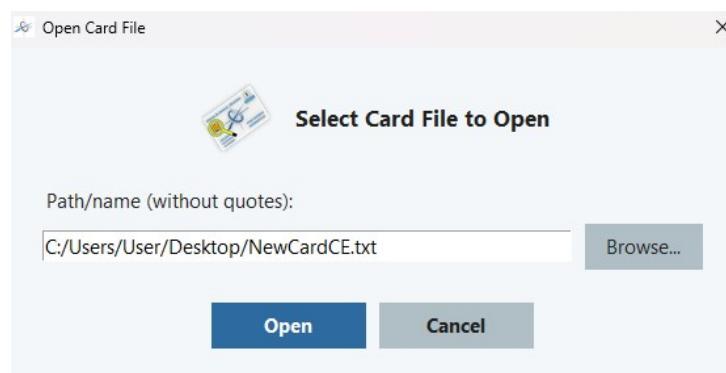


Figura 6.6. Diálogo para abrir tarjetas previamente guardadas

El diálogo de apertura (Figura 6.6) permite cargar tarjetas guardadas en sesiones anteriores. Al hacer clic se abre un navegador de archivos del sistema operativo donde el usuario selecciona el archivo .txt a importar.

Implementación técnica: El proceso de carga está gestionado por el método `open_card_from_file()` de SessionManager, que implementa un parser inteligente que detecta automáticamente el tipo de tarjeta analizando tanto los metadatos del header como el tamaño del volcado de memoria. Esta detección automática elimina la necesidad de que el usuario especifique manualmente el tipo, reduciendo errores potenciales. El parser extrae los bytes hexadecimales línea por línea, validando el formato y descartando comentarios y separadores visuales, para finalmente reconstruir el array de memoria completo mediante MemoryManager. Un aspecto crítico del proceso de carga es la sincronización posterior del estado entre MemoryManager y APDUHandler, asegurando que valores como el contador de errores y el PSC interno queden correctamente establecidos tras la importación.

La tarjeta importada restaura completamente su estado previo:

- Contenido íntegro de la memoria
- PSC actual (de fábrica o modificado)
- Valor del contador de errores
- Estado de todos los Protection Bits

Esta funcionalidad permite continuar trabajando con tarjetas de sesiones anteriores, compartir configuraciones entre usuarios, o mantener una biblioteca de tarjetas de ejemplo para prácticas educativas.

Botón Save Card - Guardar tarjeta virtual en formato .txt

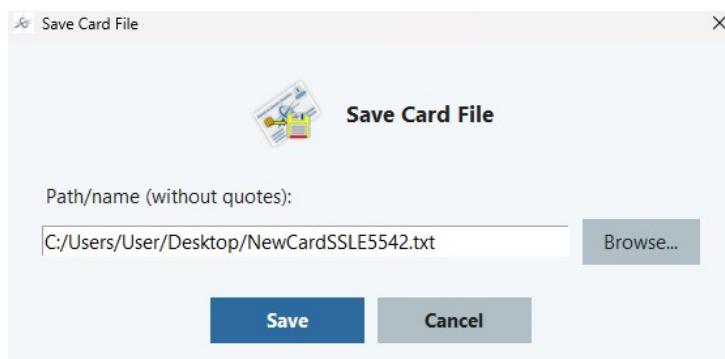


Figura 6.7. Diálogo para guardar tarjetas en archivo

La función de guardado (Figura 6.7) exporta la tarjeta actualmente seleccionada a un archivo. El usuario especifica la ubicación en el sistema de archivos y el nombre del archivo (por defecto se sugiere el nombre de la tarjeta con extensión .txt).

Implementación técnica: El método `save_session_to_file()` de `SessionManager` genera un archivo de texto estructurado que incluye metadatos en formato de comentarios (tipo de tarjeta, fecha de creación, estado del PSC, información de usuario configurada en `UserConfig`) seguidos del volcado de memoria organizado en filas y columnas hexadecimales con su representación ASCII correspondiente. Para tarjetas SLE5528, el formato incluye secciones separadas por páginas, reflejando visualmente la organización física de

la memoria del chip. Este formato es human-readable, facilitando inspección manual y depuración.

El archivo generado incluye:

- Volcado completo de memoria
- PSC actual
- Estado del contador de errores
- Configuración de Protection Bits
- Marca temporal de creación y última modificación
- Información de autoría (si está configurada)

Recordatorio importante: El guardado es manual y debe realizarse explícitamente. Se recomienda establecer un hábito de guardado frecuente, para no perder el progreso realizado en las prácticas de la asignatura.

Botón Close Card - Cerrar tarjeta

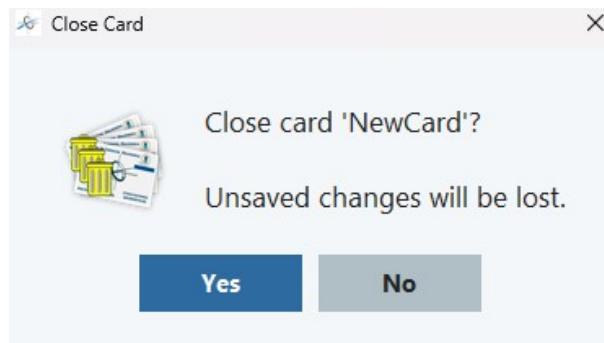


Figura 6.8. Confirmación antes de cerrar una tarjeta

Al cerrar una tarjeta (Figura 6.8), CardSIM elimina la tarjeta de la sesión actual. Se muestra una advertencia solicitando confirmación al usuario, para que pueda revisar si ha guardado el contenido previamente.

Implementación técnica: El cierre es gestionado por el método `close_session()` de SessionManager, que elimina la instancia de CardSession del diccionario

interno y limpia los archivos temporales de auto-guardado creados en el directorio temporal del sistema.

Cada CardSession mantiene su propio identificador UUID único y archivos JSON de respaldo que se actualizan automáticamente tras cada operación mediante `save_session_state()`, proporcionando red de seguridad contra cierres inesperados.

Una vez cerrada, la tarjeta desaparece del panel **Open Cards** y su contenido se pierde permanentemente de la sesión a menos que haya sido guardada previamente. Esta operación es útil para liberar espacio visual cuando se trabaja con múltiples tarjetas, reiniciar el trabajo descartando cambios no deseados, o mantener organizada la sesión de trabajo.

Botón Clear Card - Restaurar tarjeta a estado de fábrica

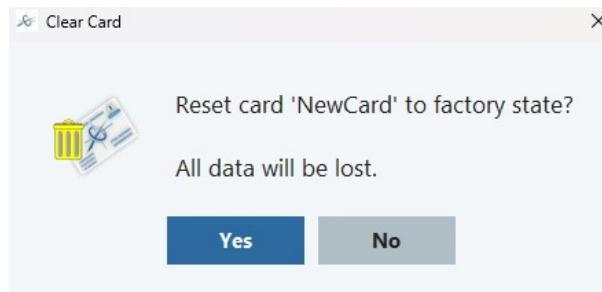


Figura 6.9. Confirmación antes de restaurar tarjeta a estado de fábrica

El botón Clear Card (Figura 6.9) restaura la tarjeta activa actual a su estado de fábrica, restableciendo todos los valores a su configuración original:

- **Memoria principal:** todos los bytes se inicializan a $0\times FF$ (255 decimal)
- **PSC (Código de Seguridad):** restaurado a FF FF o FF FF dependiendo del tipo de tarjeta (valor de fábrica)
- **Contador de errores:** reiniciado al valor máximo (7 para SLE5542, 3 para SLE5528)
- **Bits de protección:** establecidos a FF (todas las direcciones desbloqueadas)

Importante: Esta operación es irreversible y requiere confirmación explícita del usuario. Una vez ejecutada, todo el contenido previo de la tarjeta se pierde permanentemente. La tarjeta permanece abierta en la sesión (panel **Open Cards**) pero con su contenido completamente reiniciado. Si se desean conservar los datos actuales, debe guardarse la tarjeta antes de ejecutar Clear Card.

Esta función es útil para reutilizar una tarjeta virtual existente sin necesidad de cerrarla y crear una nueva, ahorrando pasos en el flujo de trabajo cuando se desea comenzar desde cero.

Botón User Info - Configuración de usuario

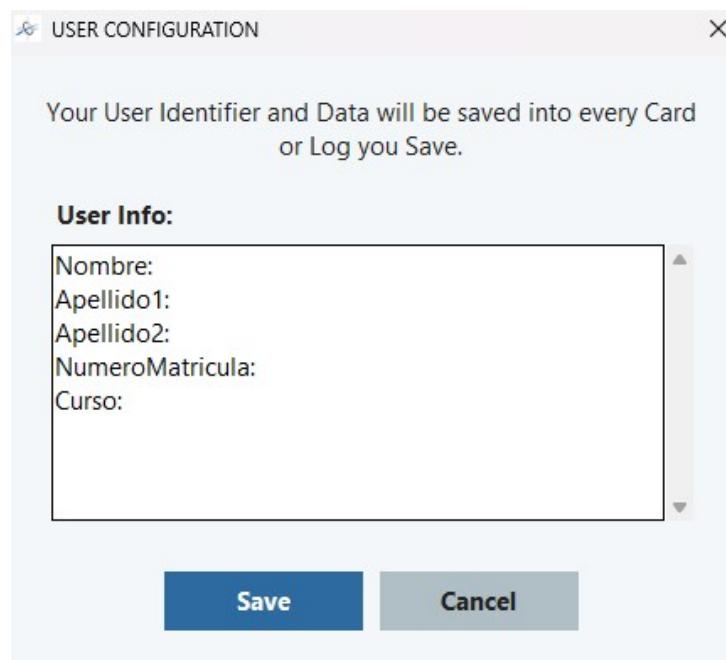


Figura 6.10. Diálogo de configuración de usuario

El diálogo de configuración de usuario (Figura 6.10) permite establecer información de identificación que se registrará en todas las tarjetas virtuales y archivos de log exportados, proporcionando trazabilidad y evidencia de autoría en entornos educativos.

El usuario puede configurar:

- **Nombre:** Nombre del estudiante

- **Apellido1:** Primer apellido
- **Apellido2:** Segundo apellido
- **Número de matrícula:** Identificador único del estudiante
- **Curso:** Asignatura específica (configurado por defecto para Comercio Electrónico)

6.3.3. Panel de comandos APDU

El panel de comandos APDU proporciona acceso directo a los 8 comandos principales definidos en la norma ISO 7816-4 para tarjetas de memoria. Aunque se trata de tarjetas virtuales, CardSIM mantiene el mismo protocolo de comunicación que se utilizaría con hardware real, proporcionando una experiencia educativa auténtica que prepara a los estudiantes para trabajar con dispositivos físicos.

Implementación técnica: La clase APDUMHandler del módulo Core constituye la implementación del subconjunto del protocolo APDU (Application Protocol Data Unit) definido en la norma ISO/IEC 7816-4 específico para tarjetas de memoria de 2 y 3 cables. A diferencia de las tarjetas con procesador que implementan el protocolo completo con selección de aplicaciones y comandos complejos, las tarjetas SLE5542 y SLE5528 utilizan un conjunto reducido de 8 comandos fundamentales que el handler procesa exhaustivamente.

Cada comando APDU sigue una estructura rígida: un byte de clase (CLA), un byte de instrucción (INS), dos bytes de parámetros (P1 y P2), y opcionalmente un byte de longitud (Le) seguido de datos (Data). El handler parsea estos componentes, valida su sintaxis, verifica que los parámetros estén dentro de rangos permitidos, y delega la ejecución a MemoryManager cuando involucran acceso a memoria. La generación de respuestas APDU (R-APDU) sigue el formato estándar, donde cada respuesta termina con un Status Word de dos bytes. CardSIM utiliza los códigos definidos en ISO 7816-4: 0x90 0x00 para operación exitosa, 0x69 0x82 para condiciones de seguridad no satisfechas, 0x67 0x00 para longitud incorrecta, y otros códigos específicos para diferentes condiciones de error. Este acercamiento al estándar permite que estudiantes que luego trabajen con hardware real reconozcan inmediatamente los patrones de respuesta.

Todos los detalles técnicos de cada comando APDU, incluyendo sus estructuras de bytes completas y responses esperadas, están documentados en el Capítulo 5 de esta memoria con tablas de referencia completas.

Máquina de estados y secuencia obligatoria

Los botones de comandos APDU se habilitan y deshabilitan dinámicamente según el estado actual de la tarjeta, implementando una máquina de estados que replica exactamente el comportamiento de las tarjetas físicas. Este mecanismo, gestionado por la clase CardSession mediante el atributo psc_verified, guía al usuario a través de las operaciones en el orden correcto y es una de las características educativas más importantes de CardSIM.

Secuencia de estados:

1. **Estado inicial:** Solo SELECT_CARD_TYPE disponible
2. **Tras SELECT:** Se habilitan operaciones de lectura y PRESENT_CODE
3. **Tras PRESENT_CODE exitoso:** Se desbloquean operaciones de escritura

| APDU Commands | |
|---------------|----------------------|
| 1 | SELECT CARD |
| 2 | READ MEMORY |
| 3 | PRESENT PSC |
| 4 | WRITE MEMORY |
| 5 | CHANGE PSC |
| 6 | READ ERROR COUNTER |
| 7 | READ PROTECTION BITS |
| 8 | WRITE PROTECT |

Figura 6.11. Estado inicial - Solo SELECT CARD TYPE habilitado

En el estado inicial (Figura 6.11), solo el comando SELECT CARD TYPE está habilitado. Este es **siempre el primer comando** que debe ejecutarse, representando el proceso de reset y activación eléctrica de la tarjeta. CardSIM lo simula

como un paso explícito para que los estudiantes comprendan que ninguna operación es posible sin activación previa.

| APDU Commands | |
|---------------|----------------------|
| 1 | SELECT CARD |
| 2 | READ MEMORY |
| 3 | PRESENT PSC |
| 4 | WRITE MEMORY |
| 5 | CHANGE PSC |
| 6 | READ ERROR COUNTER |
| 7 | READ PROTECTION BITS |
| 8 | WRITE PROTECT |

Figura 6.12. Después de SELECT - Comandos de lectura habilitados

Tras ejecutar SELECT (Figura 6.12), se habilitan los comandos de lectura (READ_MEMORY, READ_ERROR_COUNTER, READ_PROTECTION_BITS) y el comando PRESENT_CODE. Las operaciones de lectura no requieren autenticación y pueden ejecutarse libremente, permitiendo al usuario consultar el estado de la tarjeta sin riesgo.

| APDU Commands | |
|---------------|----------------------|
| 1 | SELECT CARD |
| 2 | READ MEMORY |
| 3 | PRESENT PSC |
| 4 | WRITE MEMORY |
| 5 | CHANGE PSC |
| 6 | READ ERROR COUNTER |
| 7 | READ PROTECTION BITS |
| 8 | WRITE PROTECT |

Figura 6.13. Después de PRESENT CODE - SLE5542 con escritura habilitada

| APDU Commands | |
|---------------|----------------------|
| 1 | SELECT CARD |
| 2 | READ MEMORY |
| 3 | PRESENT PSC |
| 4 | WRITE MEMORY |
| 5 | CHANGE PSC |
| 6 | READ ERROR COUNTER |
| 7 | READ PROTECTION BITS |
| 8 | WRITE PROTECT |

Figura 6.14. Después de PRESENT CODE - SLE5528 con escritura habilitada

Una vez presentado correctamente el PSC (Figuras 6.13 y 6.14), se habilitan los comandos de escritura (WRITE_MEMORY, WRITE_PROTECT, CHANGE_PSC).

Esta transición de estado solo ocurre si el PSC introducido es correcto; un PSC erróneo decrementa el contador de errores pero no desbloquea las operaciones de escritura.

A continuación se describen en detalle cada uno de los comandos APDU implementados en CardSIM, junto con sus diálogos de entrada y respuestas esperadas.

APDU 1: SELECT CARD TYPE

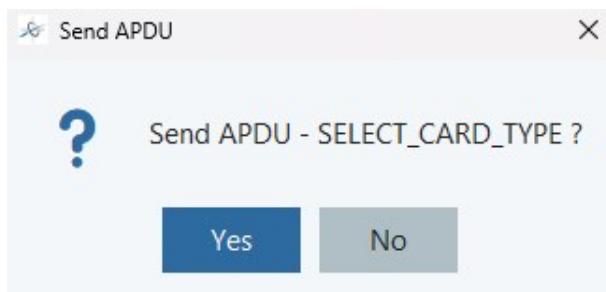


Figura 6.15. Diálogo SELECT CARD TYPE

El comando SELECT (Figura 6.15) activa la tarjeta virtual y transiciona su máquina de estados, desbloqueando las operaciones de lectura y la presentación del PSC. Aunque la tarjeta virtual ya conoce su tipo, este comando simula la operación real que debe ejecutarse en hardware físico donde el sistema host necesita identificar con qué tipo de tarjeta está comunicando.

El comando no requiere parámetros adicionales; simplemente hacer clic en el botón ejecuta la operación y muestra la respuesta en el registro de comandos, incluyendo el ATR (Answer To Reset) simulado.

Implementación técnica: El comando SELECT es procesado por APDUDHandler mediante el método `process_select_card()` que construye la estructura APDU completa según ISO 7816-4. El handler genera un ATR (Answer To Reset) simulado que identifica las características básicas de la tarjeta. Este ATR simulado permite que aplicaciones externas que intenten comunicar con CardSIM reconozcan la tarjeta. Tras la ejecución, los botones de comandos de lectura se habilitan en la interfaz mediante `update_button_states()`.

APDU 2: READ MEMORY

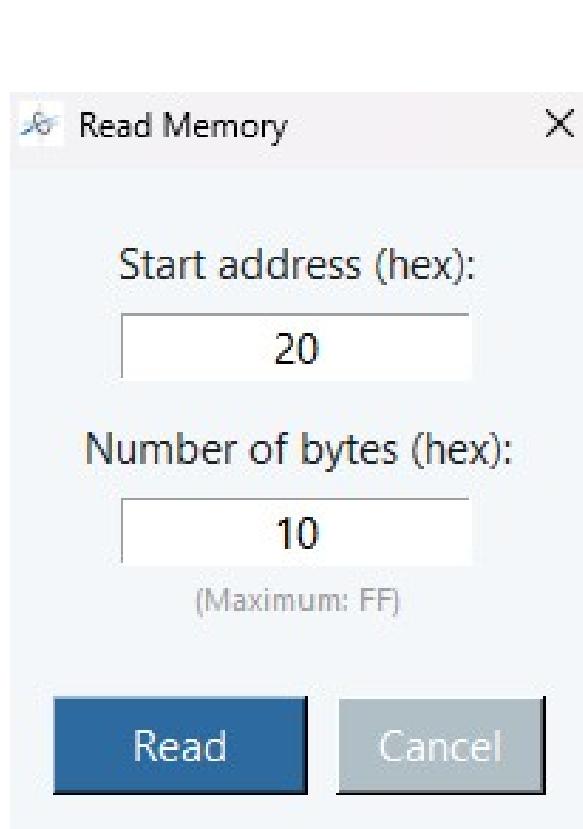


Figura 6.16. Diálogo READ MEMORY para SLE5542

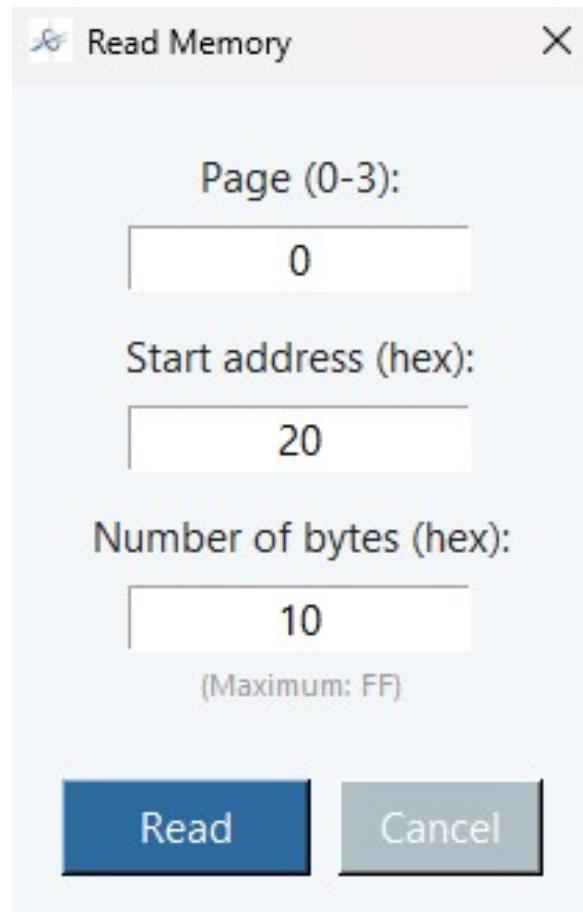


Figura 6.17. Diálogo READ MEMORY para SLE5528

El comando READ MEMORY (Figuras 6.16 y 6.17) permite consultar el contenido de un rango específico de direcciones de memoria. Al hacer clic se despliega un diálogo que solicita:

- **Dirección inicial (Address):** Dirección hexadecimal de inicio de lectura (ej: 20)
- **Longitud (Length):** Número de bytes a leer en hexadecimal (ej: 10)

Rangos válidos:

- SLE5542: Direcciones 00 a FF (256 bytes totales, una sola página)

- SLE5528: Direcciones 00-00 a 03-FF (1024 bytes totales organizados en 4 páginas: página 0 a 3, cada una con direcciones 00 a FF)

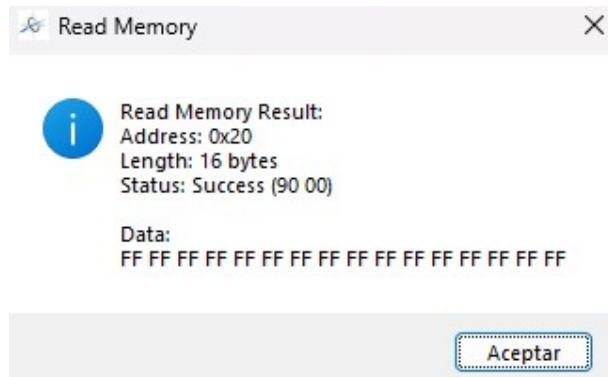


Figura 6.18. Confirmación exitosa de lectura con datos en hexadecimal

Tras la ejecución exitosa (Figura 6.18), el resultado aparece en el registro de comandos mostrando:

- El comando APDU completo enviado
 - Los datos leídos en formato hexadecimal
 - El status word de respuesta (90 00 = éxito)

Consideraciones importantes:

- No hay restricciones sobre direcciones protegidas; todas son legibles
 - La lectura no requiere autenticación previa (no necesita PSC presentado)
 - La longitud máxima de lectura está limitada por la memoria disponible desde la dirección inicial hasta el final de la página

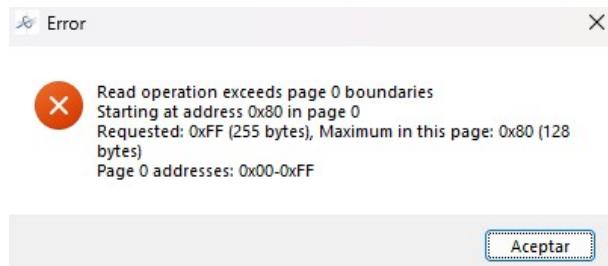


Figura 6.19. Error al intentar leer fuera del rango válido

La aplicación valida los rangos de direcciones (Figura 6.19) y previene lecturas fuera de los límites de la memoria de la tarjeta.

Implementación técnica: La lectura de memoria es ejecutada por APDUHandler mediante el método `process_read_memory()` en colaboración con MemoryManager y su método `read_memory()`. El handler primero parsea los parámetros P1 y P2 del APDU que especifican la dirección inicial y la longitud. Para SLE5542, P1 contiene la dirección completa (0x00-0xFF) mientras que para SLE5528 se combinan P1 (byte alto) y P2 (byte bajo) para formar direcciones de 10 bits (0x0000-0x03FF). El handler valida que dirección+longitud no excedan los límites de memoria específicos de cada tipo de tarjeta. Si la validación es exitosa, delega al MemoryManager que extrae los bytes solicitados del array `memory_content` y los devuelve como respuesta APDU con status word 0x90 0x00. Tras completar la lectura, el sistema de coloreado dinámico de la interfaz colorea temporalmente en cian las direcciones leídas usando el sistema de tags de Tkinter, proporcionando retroalimentación visual inmediata al usuario.

APDU 3: PRESENT CODE

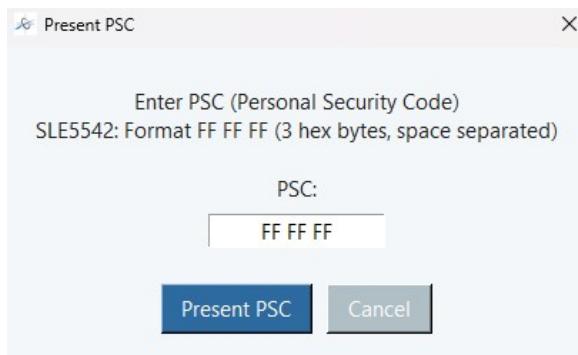


Figura 6.20. Diálogo PRESENT CODE para SLE5542 (PSC de 3 bytes)

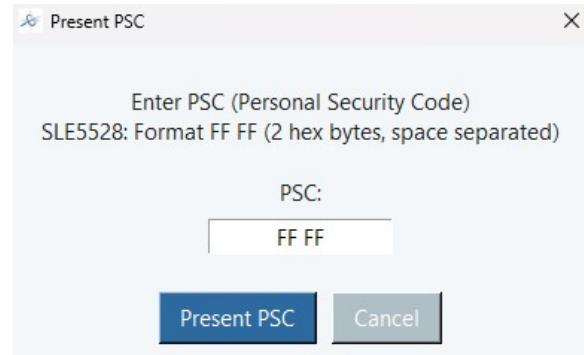


Figura 6.21. Diálogo PRESENT CODE para SLE5528 (PSC de 2 bytes)

El comando PRESENT CODE (Figuras 6.20 y 6.21) es el comando crítico que desbloquea las operaciones de escritura. Al hacer clic se muestra un diálogo solicitando el código de seguridad en el formato correspondiente:

- **SLE5542:** 3 bytes hexadecimales separados por espacios (ej: FF FF FF)

- **SLE5528:** 2 bytes hexadecimales separados por espacio (ej: FF FF)

El PSC de fábrica es FF FF FF para SLE5542 y FF FF para SLE5528. Si el usuario ha cambiado el PSC previamente mediante el comando CHANGE_PSC, debe introducir el código actualizado.

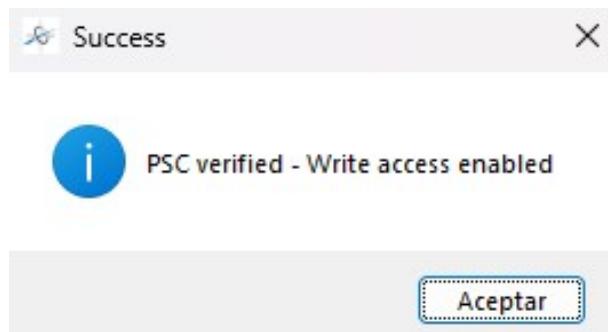


Figura 6.22. Confirmación de presentación correcta del PSC

Una presentación correcta (Figura 6.22) transiciona la máquina de estados de la tarjeta, habilitando visualmente en la interfaz los botones WRITE_MEMORY, WRITE_PROTECT y CHANGE_PSC.

Implementación técnica del sistema de seguridad: El procesamiento de PRESENT_CODE merece especial atención debido a su importancia para la seguridad de las tarjetas. El método `process_present_psc()` del APDUHandler compara el PSC proporcionado byte a byte contra el almacenado (ya sea en memoria visible para SLE5528 o en el registro interno `internal_psc_5542` del MemoryManager para SLE5542). Si la comparación es exitosa, se establece el atributo `psc_verified` de la CardSession a True. Si falla, el contador de errores se incrementa utilizando secuencias predefinidas que replican el comportamiento real de los chips.

Por ejemplo, para la SLE5542, la secuencia es 0x07 → 0x03 → 0x01 → 0x00, donde 0x00 representa el bloqueo permanente. Cuando el contador alcanza el estado bloqueado, el método `is_card_blocked()` del APDUHandler retorna True, inhabilitando todas las operaciones de escritura de forma irreversible, aunque las operaciones de lectura siguen siendo posibles.

Advertencia crítica: Introducir un PSC incorrecto suficiente veces consecutivas causa el bloqueo permanente de la tarjeta. El contador de errores se

decrementa a 0 y las operaciones de escritura quedan deshabilitadas de forma irreversible. Este comportamiento replica exactamente el mecanismo de seguridad de las tarjetas físicas y enseña a los estudiantes la importancia de manejar correctamente las credenciales de seguridad.

APDU 4: WRITE MEMORY

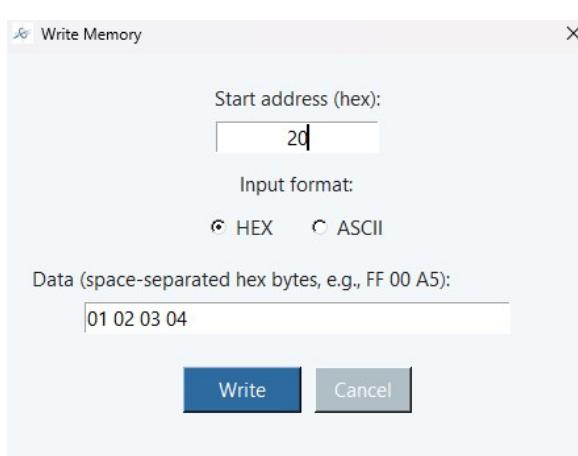


Figura 6.23. Diálogo WRITE MEMORY para SLE5542

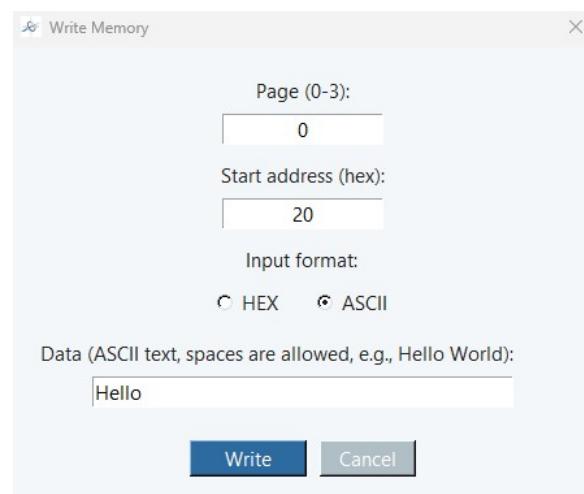


Figura 6.24. Diálogo WRITE MEMORY para SLE5528

El comando WRITE MEMORY (Figuras 6.23 y 6.24) permite escribir datos personalizados en direcciones específicas de memoria. El diálogo solicita:

- **Dirección inicial (Address):** Dirección hexadecimal de inicio de escritura
- **Longitud (Length):** Número de bytes que se escribirán
- **Formato de entrada:** Selección entre ASCII o hexadecimal
- **Datos (Data):** Los datos a escribir en el formato seleccionado

Ejemplos de uso:

- Escribir texto: Address=20, Length=05, Format=ASCII, Data=HELLO
- Escribir hex: Address=30, Length=04, Format=HEX, Data=AA BB CC DD



Figura 6.25. Confirmación de escritura exitosa

La escritura exitosa (Figura 6.25) actualiza automáticamente la visualización de memoria en el panel principal. Las direcciones escritas cambian a color **azul**, indicando que contienen datos de usuario modificables.

Restricciones importantes:

- Se permite escribir múltiples bytes en una misma operación, siempre que quepan dentro de la página destino. La comprobación consiste en verificar que la dirección inicial más la longitud de los datos no exceda el espacio disponible en la página
- No es posible escribir en direcciones protegidas (rojas en el Card Display)
- La escritura requiere PSC previamente presentado

Implementación técnica: El proceso de escritura es manejado por el método `process_write_memory()` del APDUHandler que primero verifica el estado de autenticación consultando el atributo `psc_verified` de la CardSession. Si no está autenticado, devuelve inmediatamente el status word 0x69 0x82 (condiciones de seguridad no satisfechas). Esta casuística no se debería dar, ya que existe la restricción de botones APDU Commands de la máquina de estados de la tarjeta.

Para escrituras autenticadas, el handler parsea la dirección objetivo y los datos del campo Data del APDU. Valida que la escritura no exceda los límites de la página comparando la dirección inicial con la longitud de los datos a

escribir y el espacio disponible en la página destino. Luego consulta MemoryManager mediante `is_protected()` para cada dirección objetivo rechazando escrituras en zonas protegidas.

Si todas las validaciones pasan, el método `write_memory()` del MemoryManager actualiza el array de memoria byte por byte. Finalmente, `update_card_display()` redibuja el panel completo para reflejar los cambios, aplicando coloración verde temporal a las direcciones modificadas mediante el sistema de coloreado dinámico.

APDU 5: CHANGE PSC

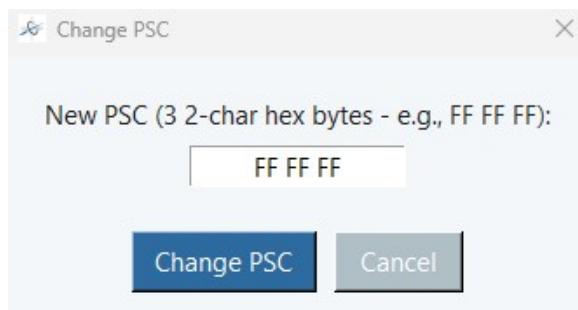


Figura 6.26. Diálogo CHANGE PSC para SLE5542

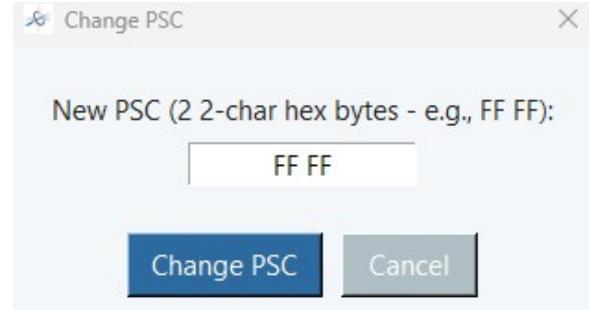


Figura 6.27. Diálogo CHANGE PSC para SLE5528

El comando CHANGE PSC (Figuras 6.26 y 6.27) permite cambiar el PSC de la tarjeta. Requiere confirmación debido a que es una operación crítica.

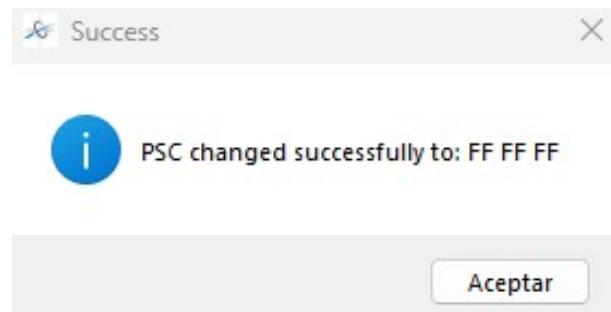


Figura 6.28. Advertencia sobre el cambio de PSC

La aplicación advierte (Figura 6.28) sobre la importancia de recordar el nuevo PSC, ya que olvidarlo bloquearía permanentemente la tarjeta.

Implementación técnica: El cambio de PSC es procesado por el método `process_change_psc()` del APDUHandler que requiere autenticación previa verificando el atributo `psc_verified` de la CardSession. El handler parsea el nuevo PSC del campo Data del APDU (3 bytes para SLE5542, 2 bytes para SLE5528) y valida que cada byte esté en el rango hexadecimal válido (0x00-0xFF).

Para la SLE5542, el nuevo PSC se almacena en el registro interno `internal_psc_5542` del MemoryManager que es un atributo separado del array de memoria principal, replicando el comportamiento del chip físico donde el PSC reside en memoria no accesible directamente. Para la SLE5528, el cambio actualiza directamente las direcciones del `memory_content` donde el PSC es visible y legible.

Tras el cambio exitoso, se actualiza el PSC interno de la sesión, accesible mediante el método `get_current_psc()` de la CardSession, para que la interfaz lo muestre correctamente en el panel de información. Se genera una entrada en el Command Log advirtiendo del cambio al nuevo código.

APDU 6: READ ERROR COUNTER

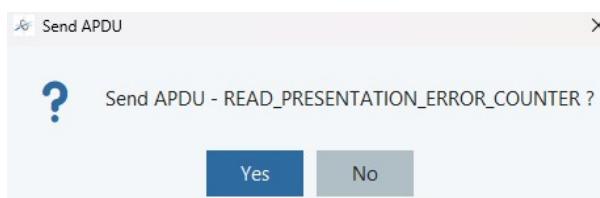


Figura 6.29. Resultado del comando READ ERROR COUNTER

El comando READ ERROR COUNTER (Figura 6.29) muestra el valor actual del contador de errores de la tarjeta.

Cuando el contador llega a 0, la tarjeta queda permanentemente bloqueada para escritura. Esta funcionalidad replica el comportamiento de las tarjetas físicas y enseña a los estudiantes la importancia de manejar correctamente las credenciales de seguridad. El comando no requiere parámetros y el resultado se muestra directamente en el registro de comandos.

Implementación técnica: El comando accede directamente al atributo `error_counter` del APDUHandler desde la interfaz. Para la SLE5542, este contador se

almacena como valor entero internamente pero se interpreta usando la secuencia de bytes específica del hardware real: 0x07 para 3 intentos restantes, 0x03 para 2 intentos, 0x01 para 1 intento, y 0x00 para tarjeta bloqueada. Para la SLE5528, el contador de errores sigue una secuencia específica (0xFF, 0x7F, 0x7E, 0x7C, 0x78, 0x70, 0x60, 0x40, 0x00) que representa los 8 intentos disponibles. El status word siempre es 0x90 0x00 ya que la lectura del contador no puede fallar aunque la tarjeta esté bloqueada.

APDU 7: READ PROTECTION BITS

El comando READ PROTECTION BITS consulta qué direcciones de memoria están protegidas contra escritura. Al hacer clic se despliega un diálogo solicitando confirmación de la operación.

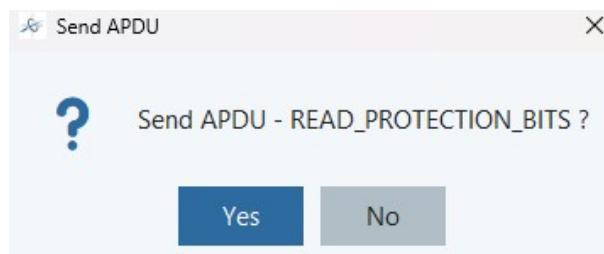


Figura 6.30. Confirmación de lectura de Protection Bits

Tras la confirmación (Figura 6.30), el comando ejecuta la lectura y muestra los resultados en una ventana de interfaz visual (Figuras 6.31 y 6.32) además de registrar la respuesta APDU completa en el Command Log.

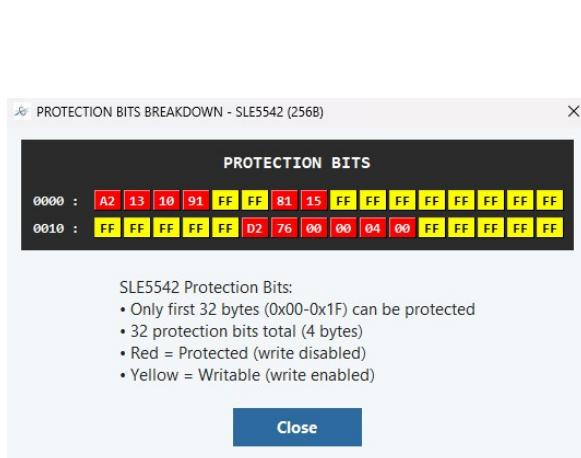


Figura 6.31. Resultado visual de Protection Bits para SLE5542

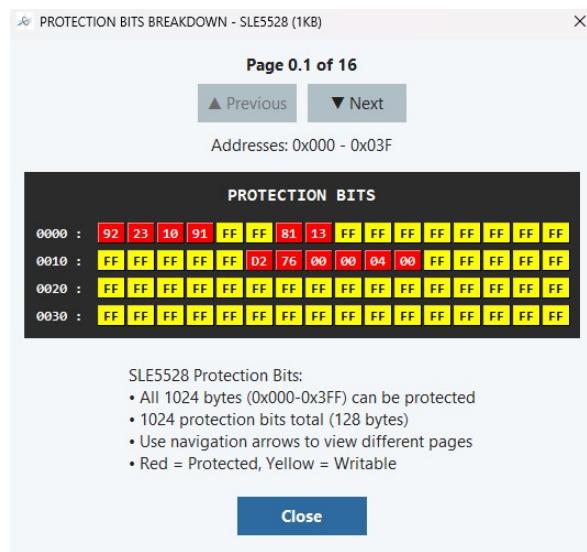


Figura 6.32. Resultado visual de Protection Bits para SLE5528

La ventana de resultados visualiza el estado de protección mediante codificación por colores:

- **Rojo:** Bits protegidos (bit a 0) - direcciones bloqueadas contra escritura
- **Amarillo:** Bits sin bloquear (bit a 1) - direcciones modificables

Diferencias entre modelos:

Para la **SLE5542** (Figura 6.31), la ventana muestra únicamente las dos primeras filas, correspondientes a los primeros 32 bytes de memoria que son los únicos direcciones protegibles en este modelo. Cada bit representa el estado de protección de una dirección específica.

Para la **SLE5528** (Figura 6.32), la ventana muestra 4 filas a la vez e incluye botones de navegación que permiten recorrer de 4 en 4 filas todas las direcciones de la tarjeta (1024 bytes totales). La interfaz indica claramente la página y sección actual en la que se encuentra el usuario durante la navegación, facilitando la comprensión de qué zona de memoria se está consultando.

Funcionalidad dual: Este comando APDU devuelve por el Command Log la respuesta completa que daría un lector real (todos los bytes de Protection Bits en formato hexadecimal seguidos del status word 0x90 0x00), simulando

el funcionamiento del hardware físico. Sin embargo, CardSIM incluye adicionalmente la ventana de interfaz visual con codificación por colores para que los estudiantes puedan comprender más fácilmente qué direcciones están protegidas sin necesidad de interpretar manualmente los bits hexadecimales.

Implementación técnica: La lectura de Protection Bits accede directamente al método `get_protection_bits()` del `MemoryManager` desde la interfaz, que consulta las direcciones específicas donde cada tipo de tarjeta almacena estos datos.

Para la SLE5542, se obtienen 4 bytes que representan los Protection Bits de los primeros 32 bytes de memoria. El método `get_protection_bits()` devuelve el array de bytes donde cada bit (1=desprotegida, 0=protegida) mapea a una dirección de memoria de usuario.

Para la SLE5528, los Protection Bits ocupan 128 bytes (direcciones 0x00-0x7F de un área protegida especial) proporcionando protección granular bit-a-bit para las 1024 direcciones. La respuesta APDU incluye todos estos bytes seguidos del status word 0x90 0x00.

APDU 8: WRITE PROTECT

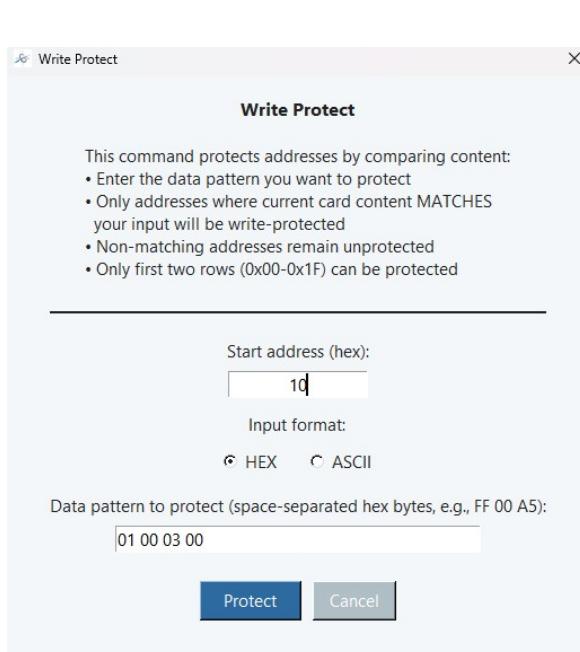


Figura 6.33. Diálogo WRITE PROTECT para SLE5542

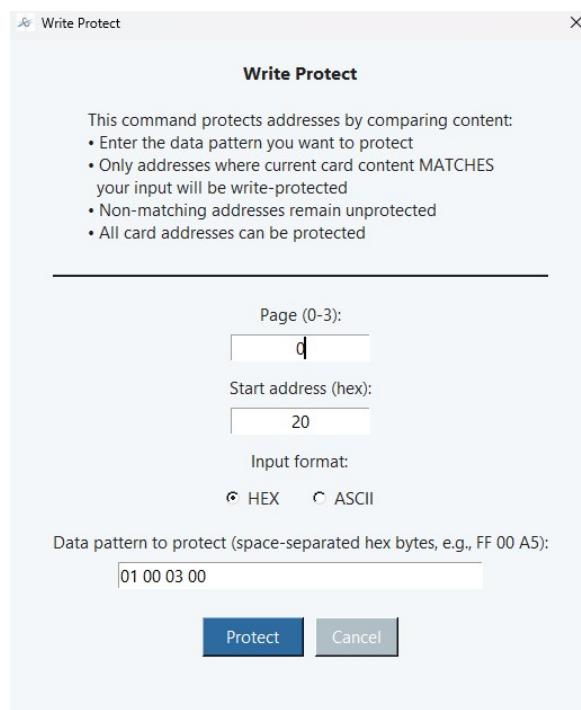


Figura 6.34. Diálogo WRITE PROTECT para SLE5528

El comando WRITE PROTECT (Figuras 6.33 y 6.34) protege permanentemente direcciones de memoria. Es una operación irreversible que requiere confirmación.

Implementación técnica: La gestión de Protection Bits está implementada en el MemoryManager, que mantiene un conjunto `protection_data` con todas las direcciones protegidas. Cuando se ejecuta una operación `WRITE_PROTECT` exitosa, la dirección correspondiente se añade a este conjunto y queda permanentemente bloqueada para escrituras futuras. Esta implementación replica el comportamiento de hardware donde los Protection Bits son irreversibles: una vez activados (cambiados de 1 a 0), no pueden revertirse. El gestor valida todas las operaciones de escritura contra este conjunto mediante el método `is_protected()`, rechazando intentos de modificar direcciones protegidas con mensajes de error apropiados. En el Card Display, estas direcciones aparecen con fondo rojo para máxima visibilidad.

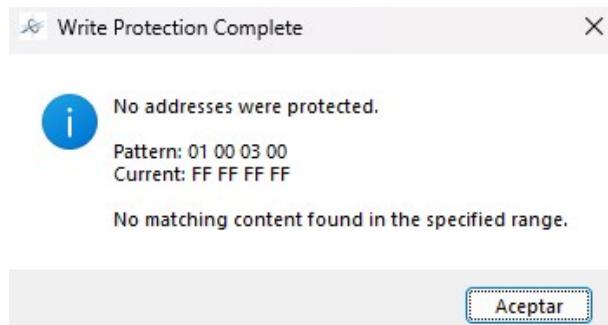


Figura 6.35. Intento de protección sin coincidencia de datos

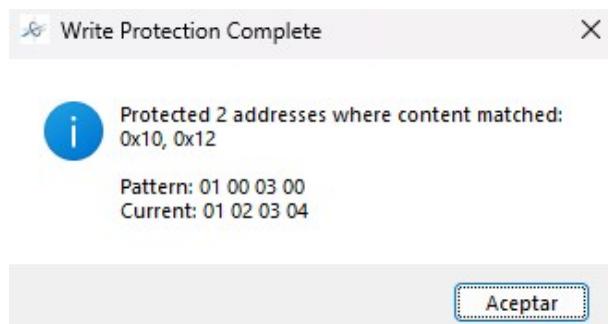


Figura 6.36. Protección exitosa de direcciones

El comando WRITE PROTECT requiere que los datos especificados coincidan con el contenido actual de la memoria (Figuras 6.35 y 6.36). Esta verificación previene protecciones accidentales.

6.3.4. Panel de información de tarjeta

Debajo de los comandos APDU (recuadro azul en la Figura 6.1), se muestra información clave de la tarjeta activa. Este panel se actualiza automáticamente mediante el método `update_card_info_display()` cada vez que cambia el estado de la tarjeta (tras ejecutar comandos APDU, presentar PSC, decrementar el contador de errores, etc.). Los datos mostrados provienen directamente de los atributos de la CardSession activa (`card_type`, `current_psc`, `error_counter`, etc.).



Figura 6.37. Panel de información para tarjeta SLE5542



Figura 6.38. Panel de información para tarjeta SLE5528

Información de tarjeta SLE5542

Para la SLE5542 (Figura 6.37) se muestra:

- **Card Name:** Nombre descriptivo de la tarjeta virtual
- **Type:** 5542 (256B) - Tipo de tarjeta y capacidad de memoria
- **Status:** Estado de la tarjeta (Created o Selected)
- **PSC:** Estado de autenticación (Not presented o Verified)
- **Error Counter:** Valor hexadecimal seguido del número de intentos restantes en decimal (ej: 0x07 (3 attempts))

Información de tarjeta SLE5528

Para la SLE5528 (Figura 6.38) la información es similar pero adaptada:

- **Card Name:** Nombre descriptivo de la tarjeta virtual
- **Type:** 5528 (1KB) - Tipo de tarjeta y capacidad de memoria
- **Status:** Estado de la tarjeta (Created o Selected)
- **PSC:** Estado de autenticación (Not presented o Verified)
- **Error Counter:** Valor hexadecimal seguido del número de intentos restantes en decimal (ej: 0x7F (7 attempts))

6.3.5. Panel de páginas

Este panel muestra todas las tarjetas virtuales actualmente cargadas en la aplicación, permitiendo trabajar con múltiples sesiones simultáneamente. Es gestionado por el componente CardExplorer, que mantiene una representación visual de todas las sesiones activas administradas por el SessionManager. Cada tarjeta se muestra como un botón interactivo creado dinámicamente mediante el método `add_card()`.

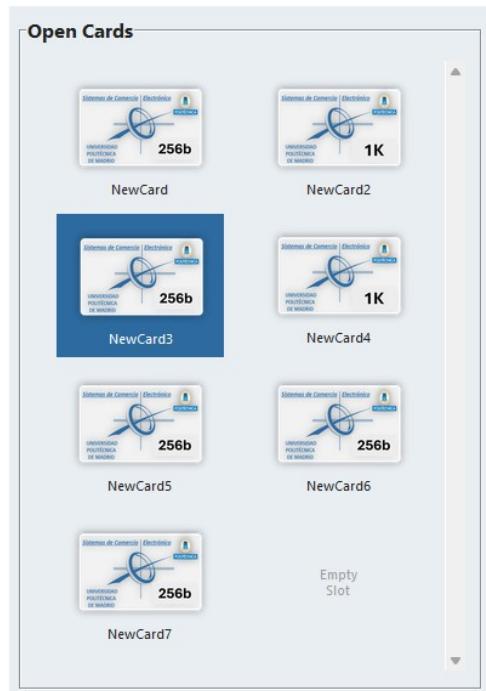


Figura 6.39. Panel con múltiples tarjetas abiertas

El panel de tarjetas abiertas (Figura 6.39) muestra cada tarjeta con su nombre descriptivo. La tarjeta activa se resalta visualmente mediante un borde de color distintivo, y un simple click sobre cualquier tarjeta invoca el método `set_active_session()` del SessionManager para cambiar el contexto de trabajo.

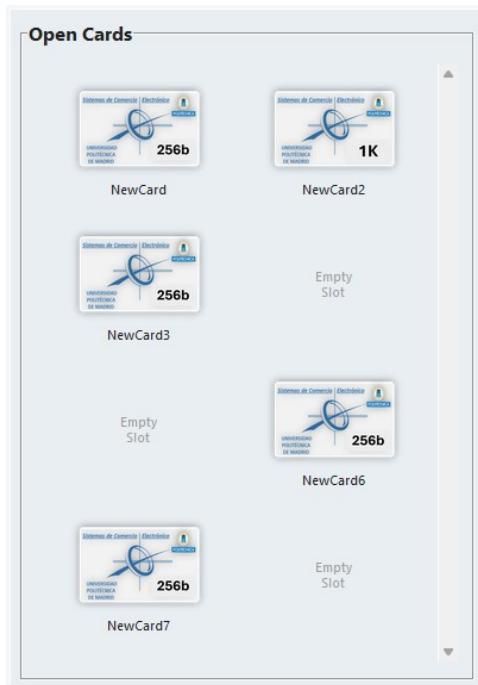


Figura 6.40. Panel después de cerrar algunas tarjetas

El panel (Figura 6.40) se actualiza dinámicamente cuando se cierran tarjetas mediante el método `remove_card()` del CardExplorer. La aplicación soporta múltiples tarjetas abiertas simultáneamente, facilitando el trabajo con diferentes configuraciones de memoria. Además, las tarjetas mantienen su posición en el panel a pesar de que se cierren otras tarjetas.

6.3.6. Panel de visualización de memoria

El panel central muestra el contenido completo de la memoria de la tarjeta activa en un formato estructurado de tres columnas. Este panel es gestionado por el método `update_card_display()`, que recibe datos directamente del MemoryManager de la sesión actual y los formatea para visualización.

El sistema de coloreado dinámico resalta visualmente diferentes zonas de memoria y resultados de operaciones:

- **Blanco:** Bytes con valor $0\times FF$ de fábrica, modificables por el usuario
- **Azul:** Direcciones modificadas por el usuario (valores diferentes a $0\times FF$)

- **Rojo:** Direcciones protegidas (protección de fábrica, protegidas por el usuario, o zonas especiales como Error Counter y PSC en SLE5528)

Estos colores se aplican mediante tags de Tkinter y se actualizan automáticamente tras cada comando APDU ejecutado, proporcionando retroalimentación visual inmediata.

Visualización de memoria - SLE5542

| Card Memory Content | | Pages |
|---------------------|--|-------|
| ADDR | HEX CONTENT | |
| 00 | 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F | P0 |
| 10 | A2 13 10 91 FF FF 81 15 FF FF FF FF FF FF FF FF FF | |
| 20 | FF FF FF FF D2 76 00 00 04 00 FF FF FF FF FF FF FF | P1 |
| 30 | 01 02 03 04 FF | |
| 40 | FF | |
| 50 | FF | |
| 60 | FF FF FF FF 48 65 6C 6C 6F 57 FF 72 6C 64 FF | |
| 70 | FF | |
| 80 | FF FF FF FF FF FF 54 53 49 SF 54 46 47 FF | |
| 90 | FF | |
| A0 | 41 42 43 44 45 46 47 48 FF FF FF FF FF FF FF FF FF | |
| B0 | FF | |
| C0 | FF | |
| D0 | FF | |
| E0 | FF | |
| F0 | FF FF FF FF FF FF 54 45 53 54 FF FF FF FF FF FF FF | |
| | | P2 |
| | | P3 |

Figura 6.41. Visualización de memoria para tarjeta SLE5542 (256 bytes)

Para la SLE5542 (Figura 6.41), el panel muestra los 256 bytes (direcciones 0000-0OFF) en tres columnas:

- **ADDR:** Dirección en hexadecimal de 2 dígitos, indica la fila
- **HEX:** Contenido del byte en formato hexadecimal de 2 dígitos, indica la columna
- **ASCII:** Representación ASCII si el byte es imprimible (punto "." si no lo es), con indicación de columna también

Implementación técnica: El MemoryManager proporciona una abstracción de alto nivel sobre el array de bytes que simula la EEPROM del chip. La inicialización mediante el método `initialize_memory()` establece una configuración de fábrica realista que replica el estado de chips físicos recién salidos de producción. Para la SLE5542, esto incluye:

- Las primeras 32 direcciones (0x00-0x1F) contienen datos de configuración pregrabados de fábrica: Application ID (A2 13 10 91), Chip Code del fabricante y otros parámetros de funcionamiento, tal como se describe en el mecanismo de protección
- Estas direcciones de configuración vienen protegidas de fábrica (bit de protección a 0), siendo inmutables
- El resto de la memoria (direcciones 0x20-0xFF) se inicializa a 0xFF, disponible para escritura por el usuario
- Contador de errores en 0x07 (representando 3 intentos disponibles)

Un aspecto importante es la diferenciación entre el PSC almacenado en memoria visible frente a los registros internos. Para la SLE5542, el PSC reside en un registro interno del chip no accesible mediante lecturas normales de memoria (característica de seguridad del hardware real). CardSIM replica esto manteniendo `internal_psc_5542` como atributo separado del array de memoria principal, invisible en operaciones READ_MEMORY pero utilizado en las validaciones de PRESENT_CODE.

Visualización de memoria - SLE5528

| Card Memory Content | | Pages |
|---------------------|---|-------|
| ADDR | HEX CONTENT | |
| ROW\COL | 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F | P0 |
| 30 | FF | P1 |
| 31 | FF | P2 |
| 32 | 3D 20 50 61 67 65 20 33 20 54 45 53 54 20 3D FF | P3 |
| 33 | FF | |
| 34 | 48 65 6C 6C 6F FF | |
| 35 | FF | |
| 36 | 50 52 45 53 45 4E 54 45 44 20 50 53 43 20 34 20 | |
| 37 | 54 49 4D 45 53 20 57 52 4F 4E 47 FF FF FF FF FF | |
| 38 | FF | |
| 39 | FF | |
| 3A | FF | |
| 3B | FF | |
| 3C | FF | |
| 3D | FF | |
| 3E | 50 53 43 20 43 48 41 4E 47 45 44 20 54 4F 20 41 | |
| 3F | 41 20 42 42 FF FF FF FF FF FF FF FF FF 78 AA BB | |
| Current PSC | | |
| AA BB | | |
| Remaining Errors | | |
| 78 (4) | | |
| Reset Counter | | |
| Reset Error Counter | | |

Figura 6.42. Visualización de memoria para tarjeta SLE5528 (1024 bytes - Página 3 activa)

Para la SLE5528 (Figura 6.42), dado que tiene 1024 bytes de memoria, el panel muestra únicamente la tercera página activa (256 bytes). La navegación entre páginas se realiza mediante los botones P0-P3.

Organización de memoria SLE5528: El MemoryManager oculta las diferencias de organización entre modelos. La SLE5528 tiene 1024 bytes organizados en 4 páginas de 256 bytes cada una. La inicialización replica el estado de fábrica con las primeras 32 direcciones (0x00-0x1F) conteniendo datos de configuración pregrabados de fábrica (Application ID 92 23 10 91, Chip Code del fabricante y otros parámetros), protegidas de fábrica tal como se describe en el mecanismo de protección. El resto de la memoria (0x20-0x3FF) se inicializa a 0xFF disponible para el usuario, contador de errores en 0xFF (representando 8 intentos), y crucialmente, el PSC ocupa direcciones de memoria estándar (0x3FE-0x3FF), siendo legible mediante READ_MEMORY pero protegido contra escritura no autorizada. Esta diferencia fundamental con la SLE5542 (registro interno vs memoria visible) es una característica real de los chips que CardSIM replica fielmente.

6.3.7. Panel de páginas

Este panel (subpanel visible en las capturas de las figuras de visualización de memoria, como se observa en la Figura 6.42) funciona para ambos tipos de tarjeta, pero presenta comportamientos diferentes según el tipo:

- **Para SLE5542 (256 bytes):** Solo está habilitado el botón P0 (Página 0), ya que toda la memoria cabe en una única vista
- **Para SLE5528 (1024 bytes):** Se habilitan los botones P0, P1, P2 y P3 una vez ejecutado el comando SELECT CARD, permitiendo navegar por las 4 páginas de memoria de 256 bytes cada una

La memoria de la SLE5528 está organizada en 4 páginas:

- **P0 (Página 0):** Bytes 0000-00FF - Página principal, contiene datos de usuario
- **P1 (Página 1):** Bytes 0100-01FF - Datos de usuario adicionales

- **P2 (Página 2):** Bytes 0200-02FF - Más datos de usuario
- **P3 (Página 3):** Bytes 0300-03FF - Última página, incluye Contador de Errores (3FD) y zona de PSC (3FE-3FF)

Al hacer click en cualquier botón de página, el panel de visualización de memoria se actualiza automáticamente para mostrar los 256 bytes correspondientes a esa página. La página activa se resalta visualmente con un color diferente para facilitar la navegación.

Este sistema de páginas permite trabajar cómodamente con los 1024 bytes totales de la SLE5528 sin sobrecargar la interfaz, manteniendo la misma estructura de visualización que se usa para la SLE5542.

6.3.8. Panel de log de comandos

El panel de log (recuadro naranja en la Figura 6.1) registra cronológicamente todas las operaciones realizadas durante la sesión, proporcionando un histórico completo y detallado de la interacción con las tarjetas.

Implementación técnica: El sistema de logging está gestionado por el método `add_to_log()` de la `CardSession` que recibe mensajes desde todas las operaciones del sistema. Los mensajes se categorizan mediante el parámetro `log_type` y se aplican colores diferentes según el tipo de operación:

- **Negro (info):** Mensajes informativos generales del sistema
- **Azul y morado:** Comandos APDU enviados y texto leído en formato ASCII
- **Verde (success):** Operaciones exitosas, respuestas simuladas de lectores y datos leídos en hexadecimal
- **Amarillo (warning):** Alertas y avisos, especialmente el número de intentos restantes del contador de errores para que el usuario lo note de inmediato
- **Rojo (error):** Errores y validaciones fallidas

Cada entrada incluye automáticamente un timestamp generado por `datetime.now().strftime()`, permitiendo rastrear la secuencia temporal precisa de todas las operaciones.



The screenshot shows a "Command Log" window with three distinct sections, each representing an APDU command:

- Card session creation:** [19:32:52] [i] INFO: Card session created: NewCard_LOGTEST (5542 (256B))
APDU COMMAND
[19:32:55] [i] INFO: SELECT CARD
[19:32:55] [APDU: FF A4 00 00 01 06]
[19:32:55] [✓] SW: 90 00
- PRESENT PSC:** [19:32:57] [i] INFO: PRESENT PSC
[19:32:57] [APDU: FF 20 00 00 03 FF FF FF]
[19:32:57] [✓] SW: 90 07
- WRITE MEMORY:** [19:33:01] [i] INFO: WRITE MEMORY
[19:33:01] [APDU: FF D0 00 20 04 01 02 03 04]
[19:33:01] [✓] SW: 90 00

Figura 6.43. Log mostrando inicialización de la aplicación

Al crear una tarjeta nueva, la aplicación (Figura 6.43) muestra en el log un mensaje de bienvenida con timestamp indicando que se ha creado la sesión correctamente.

Figura 6.44. Log mostrando cambio de PSC, lectura de memoria y consulta del contador de errores

Cada operación (Figura 6.44) se registra con:

- **Timestamp:** Hora exacta en formato [HH:MM:SS]
 - **Descripción:** Texto explicativo de la operación
 - **Comando APDU:** Bytes enviados en hexadecimal (ej: FF A4 00 00 01 06)
 - **Respuesta:** Código SW1-SW2 (ej: 90 00 = Operación exitosa)

En este ejemplo se observa la ejecución de CHANGE PSC (modificando el código a AA AA AA), seguido de READ MEMORY y READ ERROR COUNTER, mostrando cómo el log captura cada comando con sus parámetros específicos.

```

Command Log

[19:34:18] ✗ ERROR: Write protection failed: SLE5542 cards can only protect addresses 0x00-0x1F (first two rows)

----- APDU COMMAND -----
[19:34:38] ⓘ INFO: WRITE MEMORY
[19:34:38] ⏺ APDU: FF D0 00 10 04 01 02 03 04
[19:34:38] ✓ SW: 90 00

----- APDU COMMAND -----
[19:34:41] ⓘ INFO: WRITE PROTECTION
[19:34:41] ⏺ APDU: FF D1 00 10 04 01 00 03 00
[19:34:41] ✓ SW: 90 00

----- APDU COMMAND -----
[19:35:01] ⓘ INFO: PRESENT PSC
[19:35:01] ⏺ APDU: FF 20 00 00 03 FF FF FF
[19:35:01] ⚠ SW: 90 03

```

Figura 6.45. Log con secuencia de escritura, protección y error de presentación de PSC

El log (Figura 6.45) documenta operaciones más complejas, incluyendo un intento fallido de protección en direcciones no permitidas (fuera de las dos primeras filas en SLE5542), escritura de datos de prueba (01 02 03 04) mediante WRITE MEMORY, protección exitosa con patrón selectivo (01 00 03 00) que protege solo las coincidencias, y finalmente un intento erróneo de PRESENT CODE usando el PSC de fábrica (FF FF FF) cuando ya se había cambiado a AA AA AA.

```

Command Log

[19:35:06] ⓘ INFO: PRESENT PSC
[19:35:06] ⏺ APDU: FF 20 00 00 03 FF FF FF
[19:35:06] ⚠ SW: 90 01

----- APDU COMMAND -----
[19:35:31] ⓘ INFO: PRESENT PSC
[19:35:31] ⏺ APDU: FF 20 00 00 03 FF FF FF
[19:35:31] ⚠ SW: 90 00

----- APDU COMMAND -----
[19:35:32] ✗ ERROR: CARD BLOCKED: 'NewCard_LOGTEST' permanently blocked - only SELECT/READ allowed

----- APDU COMMAND -----
[19:36:00] ⓘ INFO: PRESENT PSC
[19:36:00] ⏺ APDU: FF 20 00 00 03 FF FF FF
[19:36:00] ⚠ SW: 90 03

----- APDU COMMAND -----
[19:36:11] ⓘ INFO: PRESENT PSC
[19:36:11] ⏺ APDU: FF 20 00 00 03 AA AA AA
[19:36:11] ✓ SW: 90 07

```

Figura 6.46. Log mostrando bloqueo permanente de tarjeta por intentos erróneos consecutivos

La Figura 6.46 ilustra un caso crítico: intentos consecutivos de PRESENT CODE con PSC incorrecto que decrementan progresivamente el contador de errores hasta alcanzar el bloqueo permanente de la tarjeta. El log muestra claramente el mensaje de advertencia de bloqueo permanente. Posteriormente se observa la restauración del contador mediante Reset Error Counter (operación que no aparece explícitamente en el log), seguida de nuevos intentos de PRESENT CODE: uno erróneo y finalmente uno correcto con el PSC válido (AA AA AA).

Reset Error Counter

Durante las pruebas mostradas en el log anterior (Figura 6.46), se ha utilizado el botón Reset Error Counter, una funcionalidad administrativa que permite restaurar el contador de errores de tarjetas en desarrollo.

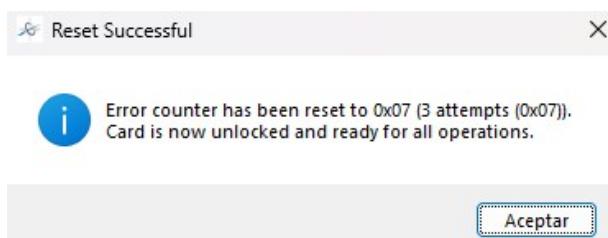


Figura 6.47. Confirmación para resetear el contador de errores

El botón Reset Counter (Figura 6.47) es una funcionalidad administrativa que permite restaurar el contador de errores cuando se ha decrementado por presentaciones incorrectas del PSC. Esta función está visible en todo momento en la interfaz para que los estudiantes puedan aprender sobre su existencia y funcionamiento.

Como se puede observar en la Figura 6.46, tras los intentos erróneos consecutivos que bloquearon permanentemente la tarjeta, el contador fue restaurado silenciosamente (sin dejar rastro en el log), permitiendo continuar con nuevos intentos de autenticación. Esta función es exclusivamente educativa y no tiene equivalente en tarjetas físicas reales, donde el bloqueo es irreversible. Su propósito es permitir a los estudiantes recuperarse de errores sin perder el progreso de sus prácticas, enfatizando al mismo tiempo la importancia de manejar correctamente las credenciales de seguridad en hardware.

real.

6.3.9. Panel de tarjetas físicas

Este panel proporciona integración con hardware real mediante lectores PC/SC compatibles.

Implementación técnica: La funcionalidad está implementada en la clase PhysicalCardHandler del módulo Core, que utiliza la librería pyscard para comunicarse con lectores y tarjetas SmartCard reales a través del estándar PC/SC (Personal Computer/Smart Card).

Durante la inicialización, el handler detecta todos los lectores PC/SC disponibles en el sistema, validando que al menos un lector esté conectado. Para cada operación física, el handler establece una conexión con la tarjeta y envía comandos APDU reales.

La construcción de APDUs físicos reutiliza exactamente la misma lógica que APDUHandler para garantizar consistencia total entre operaciones virtuales y físicas. El handler implementa manejo robusto de excepciones para capturar errores de comunicación (tarjeta no insertada, lector desconectado) y errores de autenticación PSC. Esta arquitectura transforma a CardSIM de simulador puro en herramienta híbrida, permitiendo un flujo pedagógico donde estudiantes practican primero en virtual (sin riesgo de bloqueos) y luego aplican conocimientos sobre hardware físico real con confianza.



Figura 6.48. Panel de operaciones con tarjetas físicas

El panel de tarjetas físicas (Figura 6.48) incluye dos operaciones principales:

- **Write to Physical Card:** Transfiere el contenido de la tarjeta virtual activa a una tarjeta física mediante el método `write_full_card()` del PhysicalCardHandler.

Durante la escritura, se envían las APDUs necesarias para escribir (SELECT CARD, PRESENT PSC, WRITE MEMORY), además, el procedimiento de escritura envía tantas APDUs como mitades de página se tengan que escribir. Hay que tener en cuenta que para proteger las dos primeras filas, se obvian y se comienza a escribir en la dirección $0x20$. Al finalizar, ejecuta una lectura de verificación sobre las direcciones escritas para confirmar que los datos quedaron correctamente grabados en la EEPROM.

- **Read from Physical Card:** Lee una tarjeta física y la importa como tarjeta virtual mediante el método `read_full_card()` del PhysicalCardHandler.

El handler ejecuta secuencialmente comandos READ_MEMORY sobre todo el espacio de direcciones, reconstruyendo el contenido completo de la EEPROM y permitiendo experimentación sin riesgo sobre una copia exacta del hardware.

Ambas operaciones construyen y envían comandos APDU reales al hardware, validando respuestas y manejando errores de comunicación. El PhysicalCardHandler reutiliza la misma lógica de construcción de APDUs que APDUHandler, garantizando consistencia entre operaciones virtuales y físicas. Este componente incluye manejo de las excepciones más comunes, capturando errores de comunicación con el lector (desconexión, tarjeta removida, timeouts) y situaciones de bloqueo de tarjetas (contador de errores agotado).

Lectura desde tarjeta física

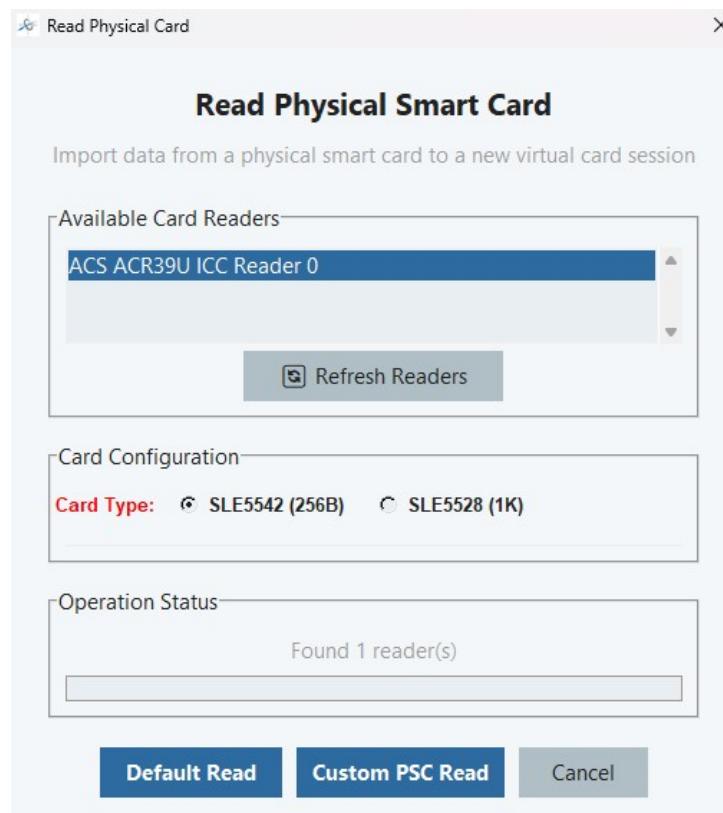


Figura 6.49. Diálogo de lectura desde tarjeta física

El diálogo de lectura (Figura 6.49) permite seleccionar el lector PC/SC conectado y el tipo de tarjeta física a leer (SLE5542 o SLE5528).

Implementación técnica: El proceso de lectura física está implementado en el método `read_full_card()` del PhysicalCardHandler. El handler primero establece conexión con la tarjeta física y detecta automáticamente el tipo

(SLE5542/SLE5528) mediante el análisis del ATR. Una vez identificado el tipo, el handler ejecuta SELECT_CARD_TYPE para inicializar la comunicación, seguido de PRESENT_CODE con el PSC proporcionado para desbloquear las zonas protegidas. Luego itera sobre todo el espacio de memoria ejecutando comandos READ_MEMORY secuenciales (256 bytes para SLE5542, 1024 bytes en 4 páginas para SLE5528), reconstruyendo el array completo de memoria. Finalmente genera una nueva tarjeta virtual con el contenido leído, permitiendo experimentación sin riesgo sobre una copia exacta del hardware físico.

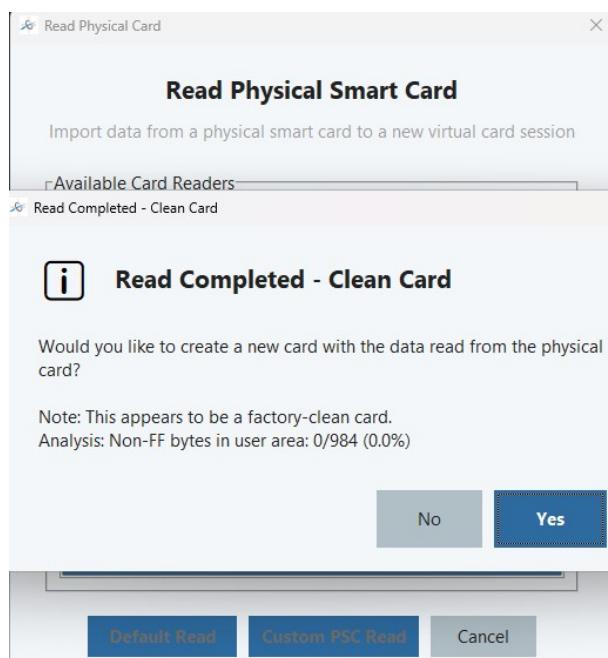


Figura 6.50. Resultado de lectura con PSC por defecto (SLE5528)

La aplicación (Figura 6.50) muestra una ventana informativa con la cantidad de contenido modificado detectado en la tarjeta física comparado con una tarjeta de fábrica, preguntando al usuario si desea crear una nueva tarjeta virtual a partir de esta lectura. El PSC de fábrica se sugiere automáticamente según el tipo de tarjeta detectado (FF FF FF para SLE5542, FF FF para SLE5528).

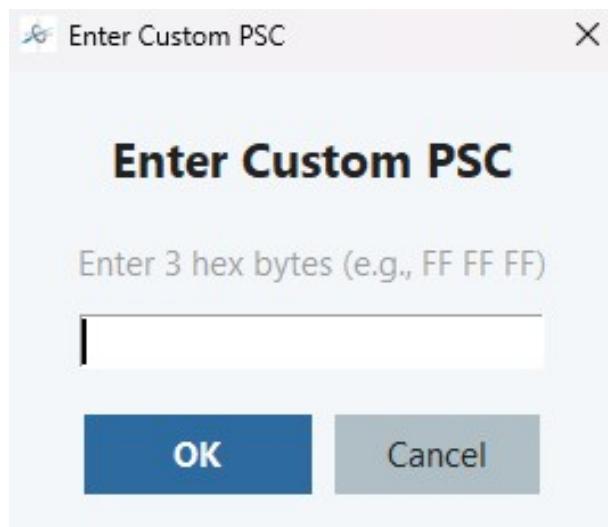


Figura 6.51. Opción para introducir PSC personalizado

Si la tarjeta física tiene un PSC modificado (Figura 6.51), el usuario puede introducirlo manualmente para acceder a las zonas protegidas. La tarjeta virtual que se cree conservará este PSC personalizado, haciendo aún más realista la aplicación de simulación de tarjetas físicas.

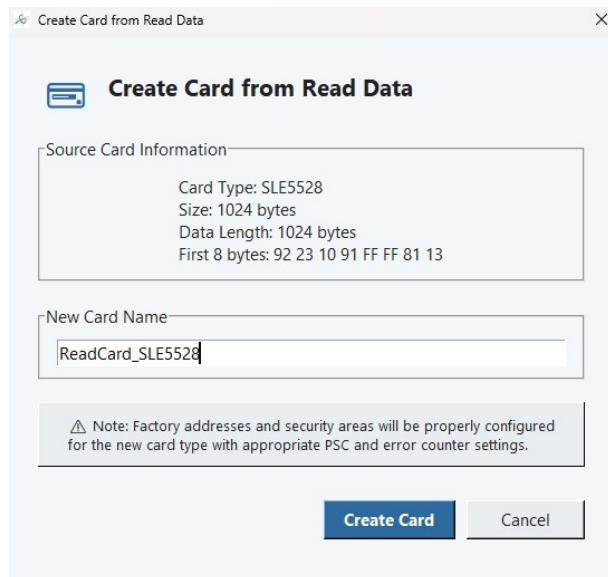


Figura 6.52. Creación de nueva tarjeta virtual desde lectura física

Tras la lectura exitosa (Figura 6.52), la aplicación crea automáticamente una nueva tarjeta virtual con todo el contenido leído desde el hardware. Se pide al usuario en la ventana de diálogo que introduzca un nombre descriptivo para

la nueva tarjeta virtual, facilitando su identificación en el panel de tarjetas abiertas.

Escritura a tarjeta física

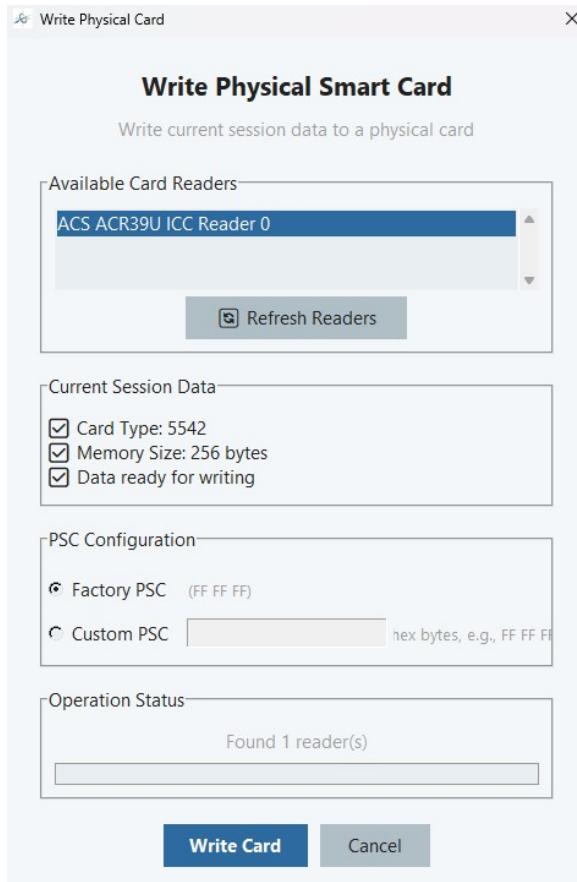


Figura 6.53. Diálogo de escritura a tarjeta física SLE5542

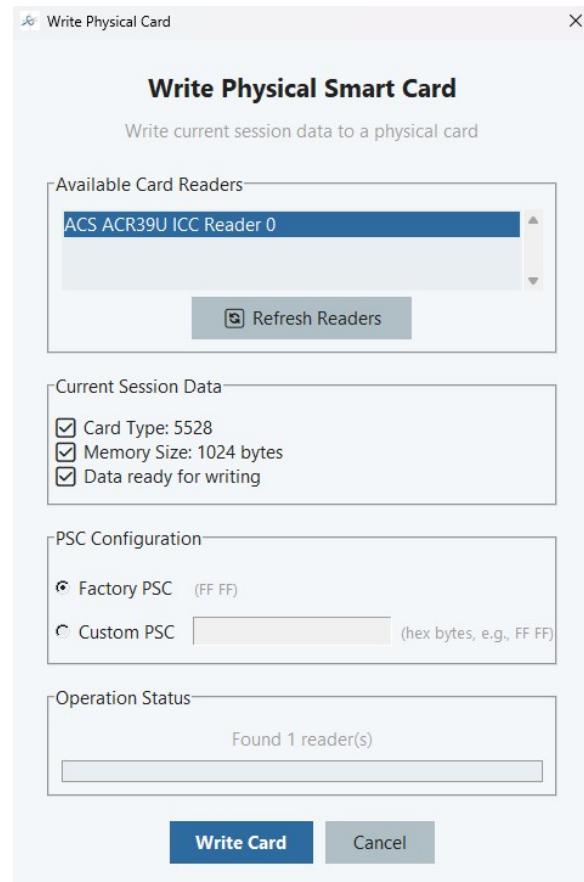


Figura 6.54. Diálogo de escritura a tarjeta física SLE5528

El proceso de escritura (Figuras 6.53 y 6.54) valida que:

- El lector esté correctamente conectado y funcional
- El PSC de la tarjeta física sea conocido (default o personalizado)

Implementación técnica: La escritura física está gestionada por el método `write_full_card()` del PhysicalCardHandler. El handler primero verifica que

existe una tarjeta virtual activa y valida la conexión física con el hardware. Ejecuta SELECT_CARD_TYPE y PRESENT_CODE en la tarjeta física para desbloquear escritura.

Para garantizar seguridad, la escritura NO se realiza en las dos primeras filas de la tarjeta (direcciones 0x00-0x1F), evitando así escrituras en zonas que puedan contener datos protegidos de fábrica. La escritura comienza a partir de la dirección 0x20.

El handler genera una secuencia de comandos APDU que se envían a la tarjeta:

- **SELECT_CARD_TYPE**: Activa la tarjeta física
- **PRESENT_CODE**: Presenta el PSC para desbloquear escritura
- **WRITE_MEMORY**: Escritura por bloques de 128 bytes para páginas completas en tarjetas de 1K. La primera página de ambos tipos de tarjeta cabe en un único comando de escritura
- **READ_MEMORY**: Lectura de verificación para comprobar que se ha escrito correctamente

Tras cada bloque de escritura, se ejecuta inmediatamente un READ_MEMORY de verificación sobre las mismas direcciones para confirmar que los datos se grabaron correctamente en la EEPROM física, comparando byte por byte contra los valores esperados. Esta verificación post-escritura detecta fallos de grabación y protecciones inesperadas.

6.3.10. Panel de acciones adicionales

Este panel agrupa funcionalidades complementarias y configuraciones avanzadas, proporcionando acceso rápido a operaciones frecuentes y herramientas de utilidad.

Implementación técnica: Las acciones adicionales están implementadas en la clase Dialogs del módulo GUI, que proporciona ventanas modales especializadas para cada funcionalidad. El sistema de diálogos utiliza `tkinter.Toplevel` para crear ventanas secundarias que bloquean la interacción con la ventana

principal hasta que el usuario complete la acción. Los diálogos de confirmación utilizan messagebox para validar operaciones destructivas (limpiar log, resetear contador de errores), mientras que los diálogos informativos (referencia APDU, créditos) emplean widgets Text de solo lectura con scroll. La exportación del log está gestionada por el método `save_log_to_file()` que utiliza `filedialog` para selección interactiva de ruta y `Text.get("1.0", "end")` para extraer todo el contenido del widget de log. El panel de configuración usa Checkbutton y Radiobutton de Tkinter vinculados directamente a atributos de UserConfig.

Panel Actions

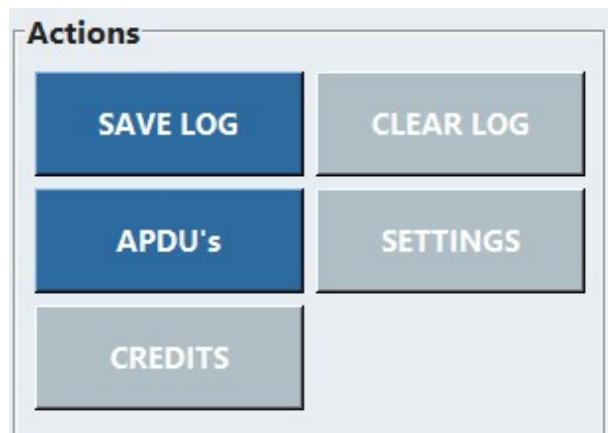


Figura 6.55. Panel Actions con botones de funcionalidades adicionales

El panel Actions (Figura 6.55) muestra una cuadrícula con diferentes botones que dan acceso a funcionalidades complementarias. A continuación se describen cada uno de estos botones en orden:

Botón Save Log - Guardar log de comandos

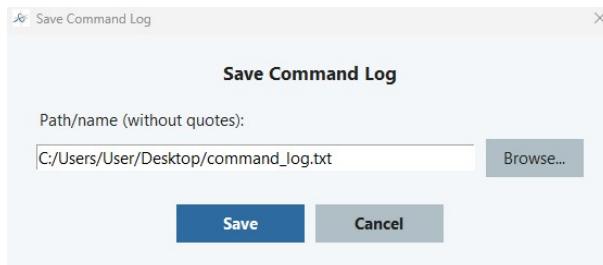


Figura 6.56. Diálogo para guardar el log de comandos

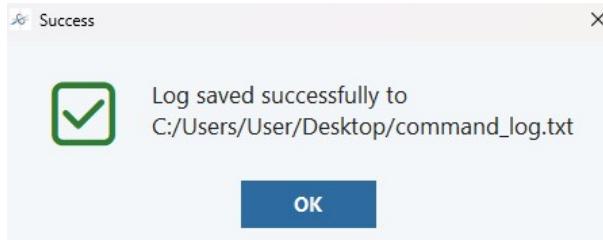


Figura 6.57. Confirmación de guardado exitoso del log

La función de guardado de log (Figuras 6.56 y 6.57) exporta todo el historial de comandos a un archivo de texto plano, útil para documentación o análisis posterior de las operaciones realizadas.

Botón Clear Log - Limpiar log de comandos

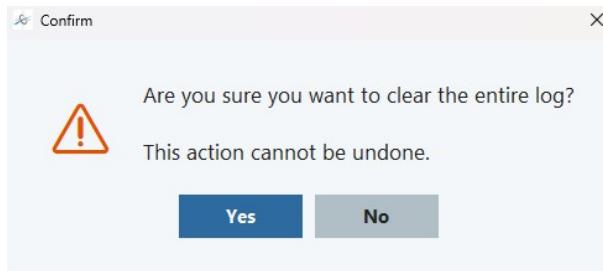


Figura 6.58. Confirmación antes de limpiar el log

La limpieza del log (Figura 6.58) requiere confirmación para evitar pérdida accidental del historial de la sesión actual.

Botón APDU's - Referencia de comandos APDU

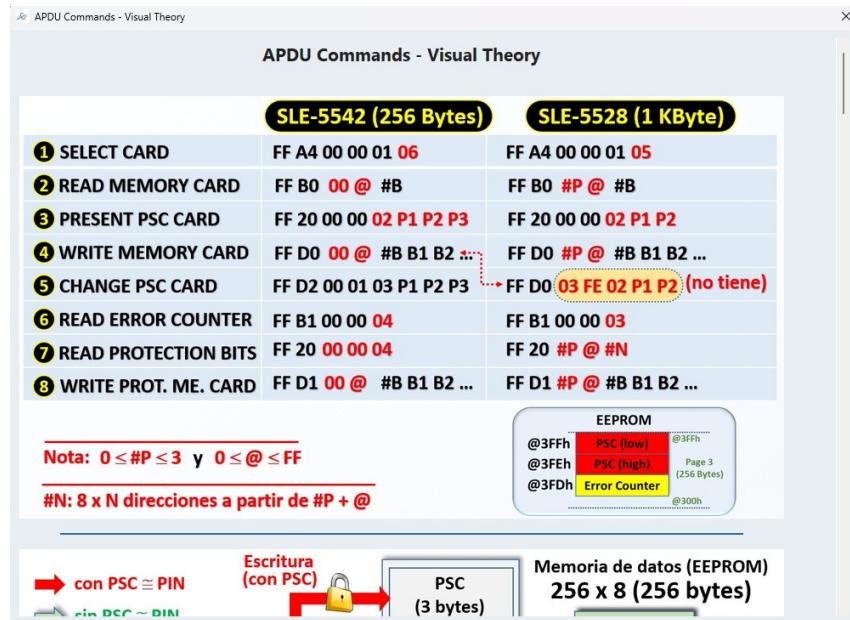


Figura 6.59. Ventana de referencia de comandos APDU

La ventana de referencia APDU (Figura 6.59) proporciona documentación integrada sobre todos los comandos disponibles, sus parámetros y formatos de respuesta según la norma ISO 7816-4.

Botón Credits - Créditos de la aplicación



Figura 6.60. Ventana de créditos y reconocimientos

La ventana de créditos (Figura 6.60) muestra información sobre el autor del proyecto, el contexto académico y los reconocimientos correspondientes.

Botón Settings - Configuración de la aplicación

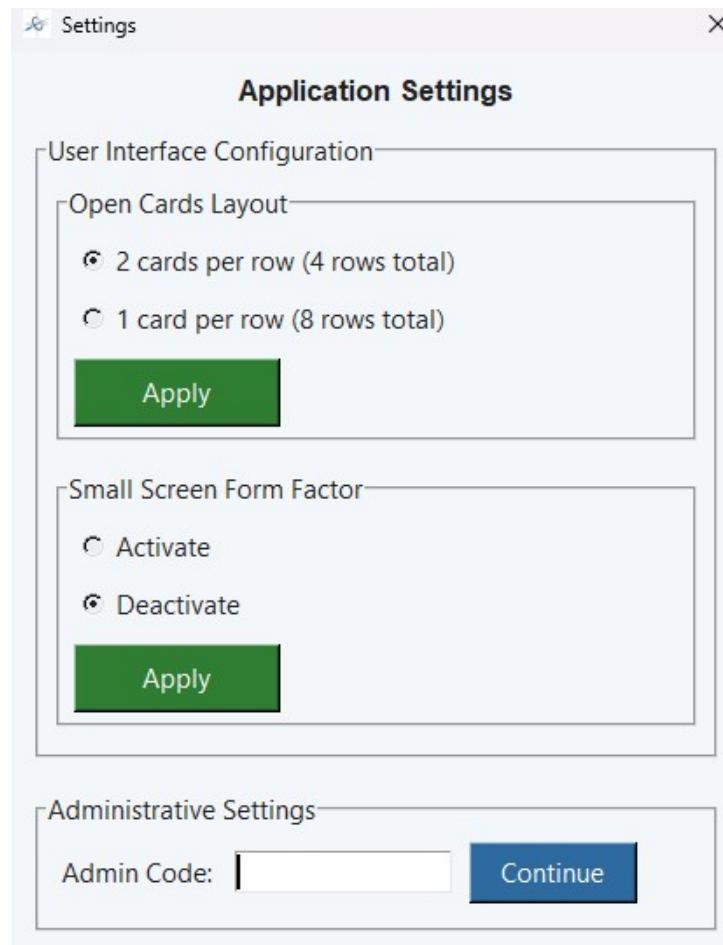


Figura 6.61. Configuración general de la aplicación

El diálogo de configuración (Figura 6.61) permite personalizar varios aspectos de la interfaz según las preferencias del usuario y las características de la pantalla. A continuación se describen las diferentes opciones de configuración disponibles.

Configuraciones de apariencia de la GUI El panel Settings permite modificar diversos parámetros visuales que afectan a la apariencia de la interfaz:

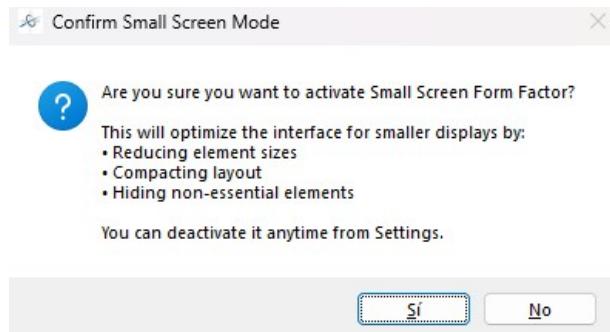


Figura 6.62. Activación del modo para pantallas pequeñas desde Settings

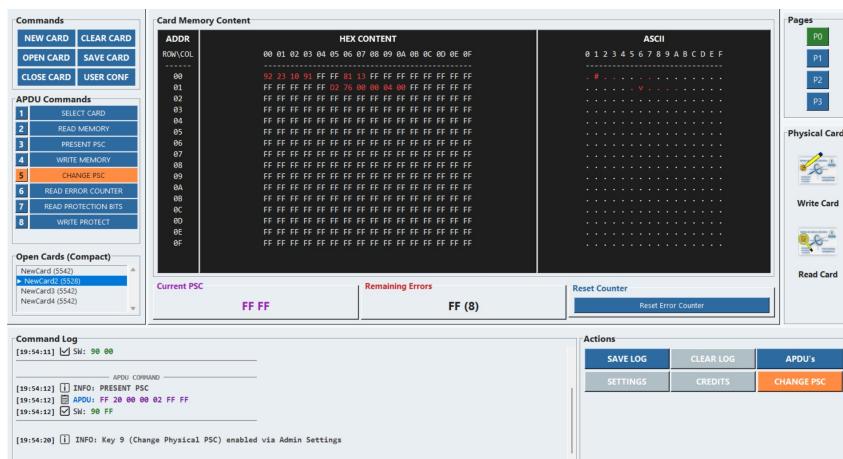


Figura 6.63. Interfaz adaptada a pantallas pequeñas

El modo Small Screen Form Factor (Figuras 6.62 y 6.63) adapta la interfaz para pantallas de menor resolución (1400x900), reduciendo el tamaño de fuentes y reorganizando elementos para maximizar el espacio útil.

CardSIM implementa adaptación automática para pantallas pequeñas (resoluciones inferiores a 1400×900 píxeles), activable desde Settings. El modo Small Form Factor realiza varios ajustes para maximizar el espacio disponible:

- El panel **Card Information** desaparece completamente, liberando espacio vertical valioso.
- El panel **Open Cards** ocupa el espacio del Card Information eliminado y simplifica su visualización, mostrando únicamente el nombre y tipo de cada tarjeta sin iconos gráficos.
- El panel **Card Memory Content** se reescaliza proporcionalmente para ajustarse al tamaño de pantalla reducido, manteniendo legibilidad.

- El panel **Physical Cards** se traslada desde su posición inferior a un sub-panel compacto debajo del panel auxiliar Pages.
- El panel **Actions** cambia su distribución de 2×3 (2 botones por fila, 3 filas) a 3×2 (3 botones por fila, 2 filas), reduciendo altura pero manteniendo todas las funcionalidades accesibles.

Esta reorganización inteligente permite usar CardSIM eficientemente en laptops con pantallas de 13-14 pulgadas sin sacrificar funcionalidad.

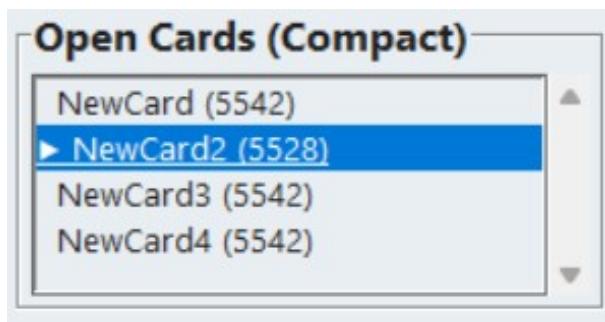


Figura 6.64. Panel de tarjetas abiertas en modo compacto

En modo compacto (Figura 6.64), el panel de tarjetas abiertas muestra 2 tarjetas por fila en lugar de 3, optimizando el uso del espacio disponible.

Además de estas adaptaciones automáticas para pantallas pequeñas, el panel Settings ofrece otras configuraciones de visualización que permiten personalizar aún más la experiencia del usuario:

Adaptación: Modo 1 tarjeta por fila:

Cuando el usuario cambia la configuración a mostrar 1 tarjeta por fila en lugar de 2 (configuración por defecto), el panel Open Cards reduce su anchura para acomodar una sola tarjeta. Este espacio liberado es aprovechado automáticamente por el panel Card Memory Content, que expande su anchura proporcionando mayor comodidad visual para lectura de contenidos de memoria extensos. Esta redistribución se gestiona dinámicamente mediante el sistema de layout de Tkinter sin requerir reinicio de la aplicación.

Otras configuraciones de apariencia:

Otras configuraciones de apariencia incluyen el cambio de colores de la inter-

faz, tamaños de fuente y distribución de paneles. Todas estas modificaciones se guardan automáticamente y se aplican en futuros arranques de la aplicación.

Sistema de adaptación de resoluciones:

CardSIM implementa dos modos de visualización para adaptarse a diferentes tamaños de pantalla:

- **Modo estándar** (Standard Screen): Optimizado para resoluciones $\geq 1536 \times 864$ píxeles. Despliega hasta 2 tarjetas por fila en el panel de tarjetas abiertas, utiliza fuentes de tamaño normal, y maximiza el espacio disponible para visualización de memoria y logs.
- **Modo compacto** (Small Screen): Diseñado para resoluciones $\leq 1400 \times 900$ píxeles. Limita a 2 tarjetas por fila, reduce el tamaño de fuentes en elementos secundarios (log, tooltips, etiquetas), compacta paddings entre widgets, y ajusta la altura de botones manteniendo los iconos. Estos ajustes permiten que la interfaz completa quepa en 900 píxeles verticales sin pérdida de funcionalidad.

Modo administrador

Como última configuración dentro de Settings, se encuentra el modo administrador. Este se activa introduciendo una clave secreta conocida únicamente por el profesor de la asignatura y el desarrollador del proyecto, desbloqueando funcionalidades avanzadas destinadas a profesores o administradores del sistema.

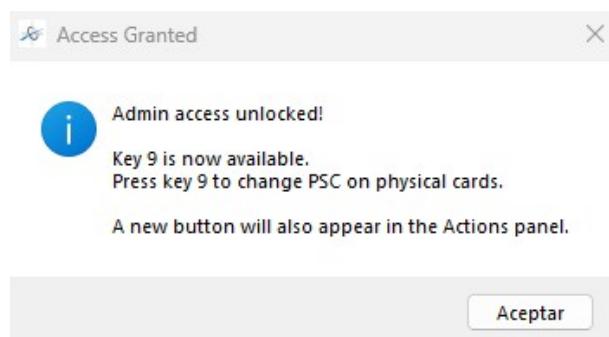


Figura 6.65. Confirmación de activación del modo administrador

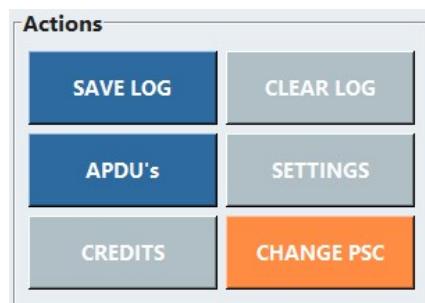


Figura 6.66. Panel con funcionalidades administrativas visibles

Una vez activado el modo administrador (Figuras 6.65 y 6.66), aparece visible el botón **Change PSC** en el panel de acciones, que permite cambiar el código de seguridad de tarjetas físicas reales.

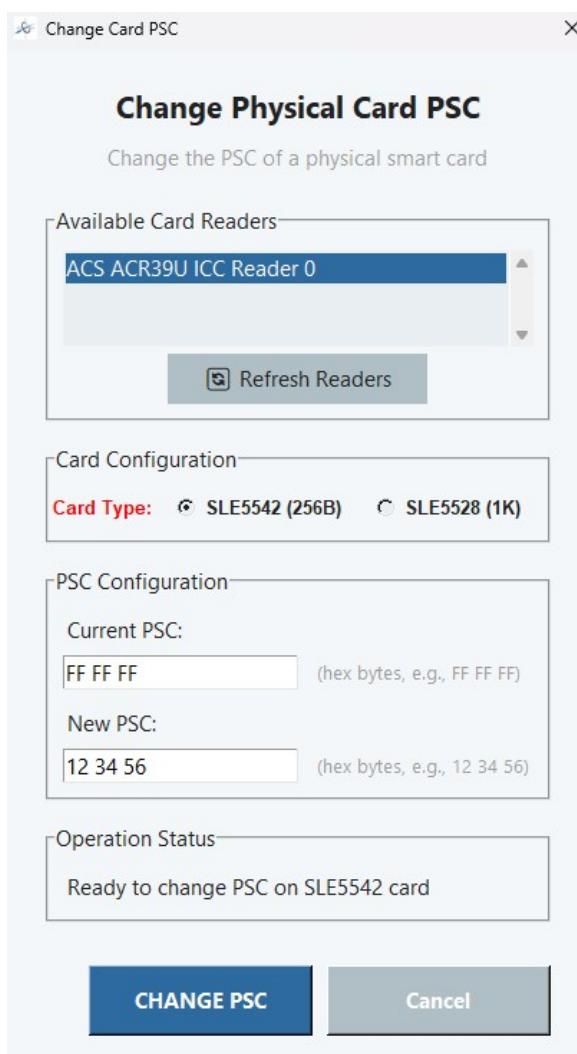


Figura 6.67. Ventana de cambio de PSC para tarjetas físicas para SLE5542

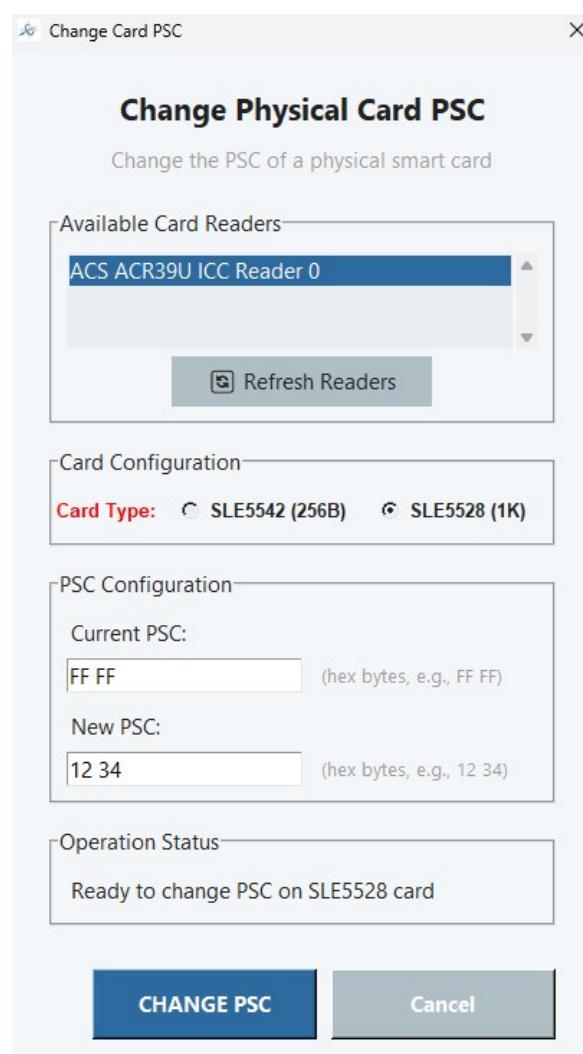
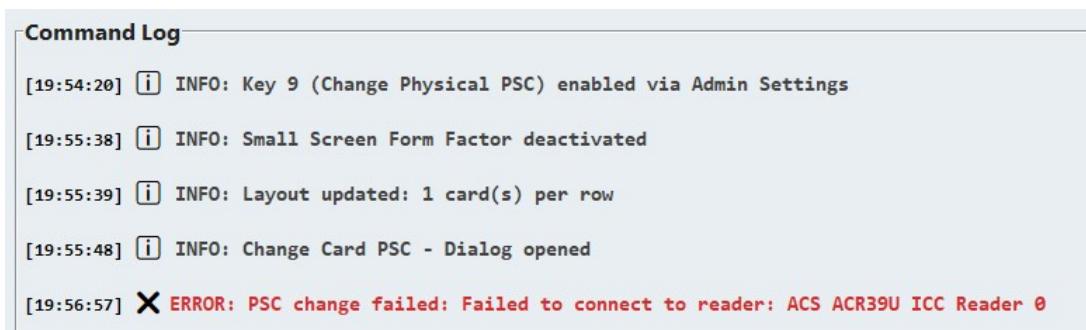


Figura 6.68. Ventana de cambio de PSC para tarjetas físicas para SLE5528

Las ventanas de cambio de PSC físico (Figuras 6.67 y 6.68) permiten modificar el código de seguridad de tarjetas reales conectadas al lector.



The screenshot shows a 'Command Log' window with the following entries:

- [19:54:20] [i] INFO: Key 9 (Change Physical PSC) enabled via Admin Settings
- [19:55:38] [i] INFO: Small Screen Form Factor deactivated
- [19:55:39] [i] INFO: Layout updated: 1 card(s) per row
- [19:55:48] [i] INFO: Change Card PSC - Dialog opened
- [19:56:57] [X] ERROR: PSC change failed: Failed to connect to reader: ACS ACR39U ICC Reader 0

Figura 6.69. Log mostrando desbloqueo de modo administrador, cambios en configuración de GUI y error de acceso al lector

En el log (Figura 6.69) se puede observar:

- El desbloqueo de la funcionalidad de modo administrador
- Los cambios realizados en la configuración de la GUI
- Un error al intentar acceder al lector de tarjetas físicas cuando no está correctamente conectado

Esta funcionalidad es especialmente útil para profesores que necesitan preparar tarjetas físicas con PSCs personalizados para prácticas específicas, o para recuperar tarjetas cuyo PSC se haya olvidado.

6.4. Flujos de trabajo típicos

CardSIM soporta dos flujos de trabajo principales que cubren diferentes escenarios de uso educativo y profesional.

6.4.1. Flujo de trabajo: Virtual a tarjeta física

Este flujo representa el caso de uso más común en entorno educativo, donde se trabaja primero en simulación y posteriormente se transfiere a hardware real:

1. **Creación de tarjeta:** El usuario crea una nueva tarjeta virtual seleccionando el tipo y asignando un nombre
2. **Selección de tipo:** Se ejecuta el comando SELECT_CARD_TYPE para confirmar el tipo
3. **Presentación de PSC:** Se presenta el código de fábrica (FF FF FF o FF FF) para habilitar escritura
4. **Escritura de datos:** Se escriben datos de usuario en las zonas configurables
5. **Verificación:** Se lee la memoria para confirmar que los datos se escribieron correctamente
6. **Protección:** Opcionalmente, se protegen direcciones críticas para evitar sobrescritura
7. **Guardado:** Se guarda la tarjeta virtual en archivo para uso posterior
8. **Escritura física:** Opcionalmente, se transfiere el contenido a una tarjeta física

Este flujo asegura que los estudiantes practiquen todos los conceptos fundamentales del manejo de SmartCards sin riesgo de bloquear hardware real.

6.4.2. Flujo de trabajo: Física a virtual y vuelta a física

Este flujo es especialmente útil cuando se necesita modificar el contenido de una tarjeta física existente sin riesgo de errores irreversibles:

1. **Lectura de tarjeta física:** Se utiliza el botón Read from Physical Card del panel Physical Cards para leer el contenido completo de una tarjeta real
2. **Selección de lector y tipo:** Se selecciona el lector PC/SC conectado y el tipo de tarjeta física (SLE5542 o SLE5528)
3. **Introducción de PSC:** Se introduce el PSC de la tarjeta física (código de fábrica o personalizado si fue modificado previamente)

4. **Creación de virtual:** La aplicación crea automáticamente una nueva tarjeta virtual con todo el contenido leído, asignándole un nombre descriptivo
5. **Modificación segura:** Se trabaja con la tarjeta virtual realizando las modificaciones necesarias (escritura de datos, protecciones, cambios de PSC)
6. **Verificación de cambios:** Se verifica mediante el panel de visualización de memoria y el log que las modificaciones son correctas
7. **Escritura a física:** Se utiliza el botón Write to Physical Card para transferir el contenido modificado de vuelta a la tarjeta física original
8. **Verificación final:** El sistema realiza automáticamente una lectura de verificación para confirmar que los datos se escribieron correctamente en el hardware

Este flujo permite actualizar tarjetas físicas de forma segura, trabajando primero en una copia virtual donde se pueden deshacer cambios si es necesario, antes de comprometerse a modificaciones permanentes en el hardware real. Es particularmente útil para profesores que necesitan preparar múltiples tarjetas físicas con configuraciones específicas para prácticas de laboratorio.

7.

Resultados, Conclusiones y Líneas Futuras

Este capítulo presenta los resultados obtenidos tras el desarrollo e implementación de CardSIM, analiza el cumplimiento de los objetivos establecidos, extrae conclusiones sobre el proyecto y propone líneas futuras de trabajo para continuar mejorando la herramienta.

7.1. Resultados

7.1.1. Implementación exitosa del sistema

CardSIM ha sido desarrollado e implementado completamente como una aplicación de escritorio funcional que cumple con todos los requisitos funcionales y no funcionales establecidos en la fase de análisis. La aplicación integra exitosamente:

- Gestión completa de tarjetas virtuales SLE5542 (256 bytes) y SLE5528 (1024 bytes).
- Implementación de los 8 comandos APDU del estándar ISO 7816-4.
- Máquina de estados que replica fielmente el comportamiento de tarjetas físicas.
- Sistema de visualización de memoria con coloreado dinámico.
- Integración con lectores PC/SC para operaciones con hardware real.
- Sistema completo de logging y exportación de operaciones.
- Funcionalidades avanzadas (modo administrador, configuración de usuario, adaptación de interfaz).

El ejecutable generado mediante PyInstaller permite la distribución y uso inmediato de la aplicación sin requerir instalación de Python ni dependencias externas, facilitando su adopción por los alumnos de la asignatura de Comercio Electrónico.

7.1.2. Evaluación en entorno académico real

La aplicación ha sido evaluada en el contexto real de la asignatura de Comercio Electrónico de la ETSISI, obteniéndose resultados altamente positivos tanto de estudiantes como del profesorado.

Feedback de estudiantes

Los estudiantes que han utilizado CardSIM durante las prácticas de laboratorio han destacado los siguientes aspectos:

- **Facilidad de aprendizaje:** La interfaz visual intuitiva ha permitido comprender rápidamente el funcionamiento interno de las tarjetas inteligentes, reduciendo significativamente la curva de aprendizaje inicial.
- **Seguridad en la experimentación:** La posibilidad de trabajar con tarjetas virtuales antes de manipular hardware real ha eliminado el miedo a cometer errores irreversibles, fomentando la experimentación activa.
- **Visualización del funcionamiento:** El coloreado dinámico de memoria y el log detallado de operaciones han facilitado enormemente la comprensión de conceptos abstractos como protecciones de memoria, códigos PSC y máquinas de estados.
- **Motivación incrementada:** El aspecto visual moderno y atractivo de la aplicación ha generado mayor interés y motivación en la realización de las prácticas comparado con herramientas de línea de comandos tradicionales.
- **Agilidad en el trabajo:** La gestión de múltiples tarjetas simultáneas y la capacidad de guardar/cargar configuraciones ha aportado gran comodidad y eficiencia al flujo de trabajo.

Evaluación del profesorado

El tutor y profesor de la asignatura ha valorado muy positivamente la herramienta, destacando:

- **Reducción drástica de bloqueos:** La tasa de tarjetas físicas bloqueadas permanentemente por errores de los estudiantes se ha reducido prácticamente a cero, eliminando uno de los principales problemas históricos de la asignatura.
- **Comprendión profunda del estándar APDU:** Los estudiantes han demostrado un entendimiento mucho más sólido de la estructura de comandos APDU y el protocolo ISO 7816-4 gracias a la implementación fiel del estándar en la aplicación.
- **Facilitación de la docencia:** La posibilidad de proyectar operaciones en tiempo real y mostrar visualmente los cambios en la memoria ha mejorado significativamente la calidad de las explicaciones en clase.
- **Herramienta educativa completa:** CardSIM no solo sirve para prácticas, sino que se ha convertido en una herramienta de referencia para demostrar conceptos teóricos durante las clases magistrales.
- **Mejora en la evaluación:** El sistema de exportación de logs permite a los estudiantes documentar exhaustivamente su trabajo, facilitando la evaluación objetiva de las prácticas realizadas.

7.1.3. Validación técnica

La aplicación ha sido sometida a pruebas exhaustivas que confirman su correcto funcionamiento:

- **Fidelidad de simulación:** Las tarjetas virtuales replican exactamente el comportamiento de las tarjetas físicas, incluyendo secuencias de contador de errores, bloqueos permanentes y protecciones irreversibles.
- **Compatibilidad hardware:** Validación exitosa con lectores ACR39U de la ETSISI, confirmando lectura/escritura correcta en tarjetas reales.

- **Estabilidad:** No se han reportado crashes o comportamientos inesperados durante el desarrollo y pruebas de la aplicación.
- **Rendimiento:** Tiempos de respuesta consistentes para operaciones APDU virtuales, proporcionando experiencia fluida.

7.2. Conclusiones

Esta sección analiza el cumplimiento de los objetivos establecidos al inicio del proyecto y extrae conclusiones generales sobre el trabajo realizado.

7.2.1. Cumplimiento de objetivos específicos

Objetivo 1: Diseñar una interfaz gráfica intuitiva y accesible

Cumplimiento: 100 %

Se ha desarrollado una interfaz gráfica completa basada en Tkinter que cumple plenamente este objetivo. La organización en paneles especializados (Commands, APDU Commands, Card Information, Memory Display, Log, Actions) proporciona acceso claro a todas las funcionalidades.

La inclusión de tooltips explicativos, iconografía distintiva, coloreado dinámico de memoria y mensajes contextuales ha logrado que estudiantes sin experiencia previa puedan operar la aplicación efectivamente en su primera sesión de uso. El feedback recibido confirma que la interfaz no solo es funcional, sino que activamente contribuye al proceso de aprendizaje.

La implementación del modo Small Screen Form Factor y la configuración flexible demuestran atención a la diversidad de entornos de uso, asegurando accesibilidad en diferentes resoluciones de pantalla.

Objetivo 2: Implementar funcionalidades de lectura y escritura segura

Cumplimiento: 100 %

CardSIM implementa todos los comandos APDU necesarios para operaciones completas de lectura y escritura en ambos tipos de tarjetas (SLE5542 y SLE5528), siguiendo rigurosamente el estándar ISO 7816-4.

Los mecanismos de validación implementados previenen efectivamente operaciones peligrosas:

- Validación de rangos de direcciones antes de operaciones de lectura/escritura.
- Verificación de límites de línea de 16 bytes en escrituras.
- Comprobación de zonas protegidas antes de permitir modificaciones.
- Confirmación explícita del usuario para operaciones irreversibles (protecciones, cambios de PSC).
- Advertencias claras sobre intentos restantes del contador de errores.
- Protección contra escrituras en zonas críticas de tarjetas físicas (primeras dos filas).

La máquina de estados implementada garantiza que los comandos solo se ejecuten cuando las condiciones de seguridad apropiadas están satisfechas (tarjeta seleccionada, PSC presentado), replicando fielmente el comportamiento de hardware real.

Objetivo 3: Integrar el soporte completo para el lector ACR39U

Cumplimiento: 100 %

La integración con lectores PC/SC mediante la librería pyscard proporciona funcionalidad completa de comunicación con hardware real. La implementación permite:

- Detección automática de lectores conectados al sistema.

- Lectura completa de tarjetas físicas SLE5528 y SLE5542.
- Escritura verificada a tarjetas físicas con validación de compatibilidad.
- Soporte para PSCs personalizados en tarjetas virtuales.
- Cambio de PSC en tarjetas físicas mediante modo administrador.
- Manejo robusto de errores de comunicación con mensajes informativos.

Aunque la especificación inicial mencionaba específicamente el ACR39U, la implementación basada en el estándar PC/SC garantiza compatibilidad con cualquier lector certificado PC/SC, proporcionando mayor flexibilidad y portabilidad de la solución.

Las pruebas realizadas en tanto en el desarrollo como en las prácticas de la asignatura de Comercio Electrónico con los lectores ACR39U confirmaron el funcionamiento sin problemas, cumpliendo plenamente el objetivo establecido.

Objetivo 4: Crear herramientas de visualización de memoria

Cumplimiento: 100 %

El sistema de visualización de memoria implementado supera las expectativas iniciales, proporcionando múltiples mecanismos para facilitar la comprensión de la estructura y organización de datos:

- **Vista estructurada:** Formato de tres columnas (Dirección, Hexadecimal, ASCII) que facilita la correlación entre representaciones.
- **Coloreado dinámico:** Sistema de códigos de color que distingue visualmente bytes de fábrica, protegidos, modificados y sin modificar.
- **Navegación por páginas:** Para tarjetas SLE5528, botones dedicados para explorar las 4 páginas de memoria.
- **Actualización en tiempo real:** Reflejo inmediato de cambios tras cada operación APDU.

- **Panel de información:** Metadatos clave (tipo de tarjeta abierta, PSC actual, errores restantes) siempre visibles.
- **Logging detallado:** Registro cronológico completo que complementa la visualización de memoria.

El feedback educativo confirma que estas herramientas de visualización han sido fundamentales para que los estudiantes comprendan conceptos complejos como la organización de páginas, zonas de seguridad, y el efecto de protecciones de memoria.

La capacidad de exportar tanto el contenido de memoria como el log de operaciones proporciona valiosa funcionalidad de documentación que contribuye al aprendizaje y evaluación por parte del profesor de la asignatura.

7.2.2. Cumplimiento del objetivo general

El objetivo general del proyecto era desarrollar una aplicación que facilitara el aprendizaje y programación segura de SmartCards SLE5528 y SLE5542, minimizando riesgos y mejorando la experiencia educativa.

Este objetivo se ha cumplido satisfactoriamente en todos sus aspectos:

- La aplicación facilita enormemente el aprendizaje mediante interfaz visual, simulación sin riesgo y herramientas de visualización.
- La programación segura está garantizada por validaciones exhaustivas y simulación previa a operaciones reales.
- El riesgo de bloqueos se ha minimizado prácticamente a cero mediante tarjetas virtuales y advertencias contextuales.
- La experiencia educativa ha mejorado significativamente según feedback de estudiantes y profesorado.

CardSIM se ha convertido en una herramienta muy valiosa para la asignatura de Comercio Electrónico, transformando la forma en que se imparten los contenidos relacionados con tarjetas inteligentes de contacto.

7.2.3. Conclusiones generales del proyecto

El desarrollo de CardSIM ha demostrado que es posible crear herramientas educativas de alta calidad que combinen rigor técnico con experiencia de usuario excepcional, incluso con recursos limitados de desarrollo individual.

Algunas conclusiones clave extraídas del proyecto:

1. **Valor de la simulación educativa:** La capacidad de experimentar sin consecuencias reales ha probado ser invaluable para el aprendizaje efectivo. Los estudiantes desarrollan confianza y competencia en entorno virtual antes de trabajar con hardware real.
2. **Importancia de la interfaz visual:** Una interfaz gráfica bien diseñada no es meramente estética, sino que activamente facilita la comprensión de conceptos complejos mediante representaciones visuales apropiadas.
3. **Fidelidad de simulación crítica:** Para que una herramienta educativa sea verdaderamente útil, debe replicar fielmente el comportamiento del sistema real, incluyendo limitaciones y restricciones. La autenticidad genera transferencia de conocimiento efectiva.
4. **Arquitectura modular como base de éxito:** La organización del código en módulos independientes (Core, GUI, Utils) facilitó enormemente el desarrollo, debugging y extensibilidad del sistema.
5. **Validación en entorno real es esencial:** Las pruebas con usuarios reales en el contexto educativo proporcionaron feedback que no podría obtenerse mediante pruebas de desarrollador.
6. **Python + Tkinter como combinación efectiva:** A pesar de ser considerados herramientas simples, Python y Tkinter demostraron ser perfectamente adecuados para desarrollar aplicaciones de escritorio profesionales cuando se utilizan apropiadamente.

7.2.4. Limitaciones identificadas

A pesar del éxito general del proyecto, se han identificado algunas limitaciones que es importante reconocer:

- **Interfaz en un solo idioma:** La aplicación está completamente en español, lo que puede limitar su uso en contextos internacionales donde no se domine el idioma.
- **Sin sistema de ayuda integrado:** Aunque la interfaz es intuitiva, no existe un sistema de ayuda contextual o documentación in-app para consultar durante el uso.
- **Rendimiento de arranque:** Se ha detectado que en algunos equipos de gama baja, la aplicación puede tardar más de lo habitual en cargar la interfaz principal.

Estas limitaciones no impiden el uso efectivo de la aplicación en su contexto objetivo, pero representan áreas de posible mejora futura.

7.3. Líneas futuras de trabajo

El éxito de CardSIM abre múltiples posibilidades de extensión y mejora que podrían abordarse en futuros trabajos:

7.3.1. Extensión de funcionalidades

- **Soporte para tipos adicionales de tarjetas:** Extender la aplicación para soportar otras familias de tarjetas inteligentes de memoria (Mifare Classic, Mifare DESFire, etc.) y potencialmente tarjetas con procesador.
- **Modo de práctica guiada:** Implementar un sistema de tutoriales interactivos paso a paso que guíen a estudiantes principiantes a través de operaciones comunes, con validación automática de progreso.
- **Sistema de escenarios predefinidos:** Crear biblioteca de ejercicios y desafíos preconfigurados que permitan al profesorado asignar tareas específicas con criterios de evaluación automatizados.
- **Modo comparativo:** Permitir visualizar dos tarjetas lado a lado para comparar contenidos y facilitar análisis de diferencias.

- **Historial de operaciones persistente:** Mantener histórico completo de todas las sesiones de trabajo con capacidad de búsqueda y análisis retrospectivo.
- **Exportación a formatos adicionales:** Además de .txt, soportar formatos estándar de la industria como .dump, .bin, o formatos específicos de herramientas comerciales.

7.3.2. Mejoras de rendimiento y optimización

- **Caché de iconos:** Optimizar carga de recursos gráficos mediante sistema de caché en memoria.
- **Reducción del tamaño del ejecutable:** Explorar alternativas a PyInstaller o técnicas de compresión para reducir el tamaño del ejecutable standalone.

7.3.3. Mejoras de interfaz y usabilidad

- **Sistema de temas:** Permitir al usuario personalizar esquema de colores (modo claro/oscuro, temas personalizados).
- **Personalización de layout:** Permitir reorganizar y redimensionar paneles según preferencias del usuario (no solo cambiar entre dos formatos predefinidos).
- **Internacionalización:** Implementar en el menú de configuraciones la opción de seleccionar idioma, facilitando su uso para cualquier usuario.

7.3.4. Funcionalidades educativas avanzadas

- **Modo simulación con latencia:** Simular tiempos de respuesta realistas de hardware para aproximar más la experiencia a la realidad.
- **Inyección de errores controlada:** Permitir al profesor configurar escenarios con errores específicos para que estudiantes practiquen troubleshooting.

- **Sistema de evaluación integrado:** Funcionalidad para que profesores creen ejercicios evaluables con criterios automáticos de corrección.

7.3.5. Aspectos técnicos y arquitectónicos

- **Suite de tests automatizados:** Desarrollar conjunto completo de tests unitarios e integración para garantizar robustez ante futuras modificaciones.
- **Documentación exhaustiva:** Crear documentación técnica completa, junto a una guía de los diferentes elementos de la aplicación, que sirviera de tutorial y primer acercamiento para nuevos usuarios.

Estas líneas futuras demuestran el potencial de crecimiento de CardSIM más allá de su implementación actual, transformándolo potencialmente en una herramienta de referencia en la enseñanza de tecnologías de tarjetas inteligentes.

8.

Bibliografía

Este proyecto se ha basado en las siguientes fuentes y referencias técnicas:

- **Estándares ISO/IEC 7816-4:2020:** Estándar internacional para comandos APDU y comunicación con tarjetas inteligentes.
- **Datasheets de Infineon Technologies:** Especificaciones técnicas completas de las tarjetas SLE5542 y SLE5528, incluyendo organización de memoria, códigos de seguridad y comandos soportados.
- **Manual del lector ACR39U:** Documentación técnica de Advanced Card Systems Ltd. para integración con lectores PC/SC.
- **Especificación PC/SC Workgroup:** Estándar para comunicación entre ordenadores personales y lectores de tarjetas inteligentes.
- **Documentación oficial de Python 3:** Referencia completa del lenguaje de programación utilizado.
- **Documentación de Tkinter:** Biblioteca estándar de Python para desarrollo de interfaces gráficas.
- **Proyecto pyscard:** Librería especializada para comunicación con SmartCards mediante PC/SC, desarrollada por Jean-Daniel Aussel y Ludovic Rousseau.
- **PyInstaller:** Herramienta para generación de ejecutables standalone desde código Python.
- **Literatura académica:** Libros de referencia sobre SmartCards (Rankl & Effing, Hendry, Guthery & Jurgensen), diseño de interfaces de usuario (Norman, Shneiderman), y patrones de diseño de software (Gamma et al., Fowler).
- **Artículos sobre educación en ingeniería:** Investigaciones sobre aprendizaje experiencial y estilos de enseñanza en ingeniería (Kolb, Felder & Silverman).

Nota: La bibliografía completa en formato BibTeX está disponible en el archivo `references.bib` del proyecto.

Este capítulo contiene documentación complementaria que amplía la información presentada en el cuerpo principal de la memoria.

9.1. Anexo A: Estructura del proyecto

La estructura completa de directorios del proyecto CardSIM es la siguiente:

```
Proyecto/
    └── main.py                                # Punto de entrada de la aplicación
    └── CardSIM.spec                            # Configuración de PyInstaller

    └── src/
        └── core/
            ├── __init__.py
            ├── card_session.py      # Gestión de sesiones
            ├── memory_manager.py   # Gestión de memoria
            ├── apdu_handler.py     # Procesamiento APDU
            ├── physical_card_handler.py # Integración con hardware
            ├── session_manager.py   # Coordinador de sesiones
            └── code_improvements.py  # Mejoras de código

        └── gui/
            ├── __init__.py
            ├── interface.py       # Ventana principal
            ├── dialogs.py         # Diálogos especializados
            ├── card_explorer.py   # Panel de tarjetas abiertas
            └── create_card_dialog.py # Diálogo creación de tarjetas
```

```
└── physical_card_dialogs.py # Diálogos tarjetas físicas  
└── utils/                  # Utilidades y helpers  
    ├── __init__.py  
    ├── app_states.py          # Estados de la aplicación  
    ├── constants.py           # Constantes globales  
    ├── resource_manager.py    # Gestión de recursos  
    └── user_config.py         # Configuración de usuario  
  
└── assets/                  # Recursos estáticos  
    └── icons/                 # Iconografía de la aplicación  
        ├── newcard.png  
        ├── opencard.png  
        ├── savecard.png  
        └── ...  
  
    └── teoria/                # Material teórico de la asignatura  
        ├── apdus.jpg  
        ├── change_psc.jpg  
        ├── present_psc.jpg  
        └── ...  
  
└── dist/                   # Ejecutables generados  
    └── CardSIM.exe            # Ejecutable Windows
```

9.2. Anexo B: Ejemplo tarjeta virtual guardada en .txt

Los archivos de tarjeta personalizados .txt utilizan un formato de texto estructurado que incluye metadatos y volcado de memoria. Ejemplo de archivo SLE5542:

```
# CardSIM Virtual Card File  
# Card Type: SLE5542
```

```
# Memory Size: 256 bytes
# Created: 2025-10-31 15:30:45
# PSC: FF FF FF
# Error Counter: 07
# User: David Balenzategui - ETSISI - 2024/2025

# Memory Dump (Hex):
0000: FF ..... .
0010: FF ..... .
0020: 48 6F 6C 61 20 4D 75 6E 64 6F FF FF FF FF FF FF FF FF FF Hola Mundo.... .
...
00F0: FF ..... .

# Protection Bits:
# [Configuración de bits de protección]
```

9.3. Anexo C: Códigos de respuesta (Status Words)

Códigos SW1-SW2 utilizados por la aplicación:

| SW1 | SW2 | Significado |
|------|------|---|
| 0x90 | 0x00 | Operación exitosa |
| 0x63 | 0x00 | Verificación fallida |
| 0x69 | 0x82 | Condiciones de seguridad no satisfechas |
| 0x6A | 0x86 | Parámetros P1-P2 incorrectos |
| 0x6B | 0x00 | Parámetros incorrectos |
| 0x6D | 0x00 | Instrucción no soportada |
| 0x6E | 0x00 | Clase no soportada |
| 0x67 | 0x00 | Longitud incorrecta |

Tabla 9.1. Status Words implementados

9.4. Anexo D: Instalación y configuración

9.4.1. Requisitos del sistema

- **Sistema operativo:** Windows 10/11
- **Memoria RAM:** Mínimo 2 GB
- **Espacio en disco:** 25 MB para la aplicación
- **Resolución de pantalla:** Mínimo 1280×720 (recomendado 1920×1080)
- **Hardware opcional:** Lector ACR39U para operaciones con tarjetas físicas

9.4.2. Instalación desde ejecutable

1. Descargar CardSIM.exe del repositorio o ubicación proporcionada
2. Ejecutar el archivo (no requiere instalación)
3. Aceptar advertencias de seguridad de Windows si aparecen
4. La aplicación se iniciará automáticamente

9.4.3. Instalación desde código fuente

```
# Clonar repositorio
git clone [URL-repositorio]
cd Proyecto

# Crear entorno virtual
python -m venv venv
source venv/bin/activate # En Windows: venv\Scripts\activate

# Instalar dependencias
pip install -r requirements.txt
```

```
# Ejecutar aplicación
python main.py
```

9.5. Anexo E: Atajos de teclado

Lista completa de atajos de teclado disponibles:

| Atajo | Acción |
|-----------------|--|
| 1-8 | Ejecutar comandos APDU 1-8 respectivamente |
| 9 | Abrir diálogo de cambiar PSC físico (si está habilitado) |
| Ctrl+1 a Ctrl+8 | Cambiar a la tarjeta abierta 1-8 |
| P+0 a P+3 | Cambiar a página 0-3 (solo SLE5528) |
| Enter | Confirmar diálogo |
| Escape | Cancelar diálogo |

Tabla 9.2. Atajos de teclado de CardSIM

9.6. Anexo F: Esquema de colores de memoria

Código de colores utilizado en la visualización de memoria:

| Color | Código RGB | Significado |
|------------|------------|----------------------------------|
| Blanco | #FFFFFF | Byte en estado de fábrica (0xFF) |
| Azul claro | #FFFFE0 | Byte modificado por el usuario |
| Rojo claro | #FFE4E1 | Byte con protección activa |
| Gris claro | #FOFOFO | Fondo general de la interfaz |

Tabla 9.3. Esquema de colores de visualización

