Universidad de Granada
Escuela Técnica Superior de Ingeniería Informática y de Telecomunicación

---

Grado en Ingeniería de Tecnologías de Telecomunicación
Especialidad en Telemática

Mecanismos de protección de datos en videojuegos

Trabajo Fin de Grado

Benito Palacios Sánchez

Tutor

Dr. D. Pedro García Teodoro
Departamento de Teoría de la Señal, Telemática y Comunicaciones

Julio de 2015

Benito Palacios Sánchez

Mecanismos de protección de datos en videojuegos

Trabajo Fin de Grado

Universidad de Granada

Yo, **Benito Palacios Sánchez**, alumno de la titulación Grado en Ingeniería de Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Benito Palacios Sánchez

Granada a 18 de diciembre de 2022.

Dr. D. **Pedro García Teodoro**, Profesor del Área de Ingeniería Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Mecanismos de protección de datos en videojuegos***, ha sido realizado bajo su supervisión por **Benito Palacios Sánchez**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que así conste, expide y firma la presente autorización en Granada a 18 de diciembre de 2022.

El director:

Dr. D. Pedro García Teodoro

Mecanismos de protección de datos en videojuegos

Benito Palacios Sánchez

Palabras clave: seguridad, videojuegos, ingeniería inversa, servicios en línea, DRM, Nintendo DS.

Resumen

Los videojuegos son cada día más un factor importante de nuestra cultura actual. Especialmente desde el *boom* que ha dado el mercado de los teléfonos inteligentes. A final de 2014 la compañía detrás del famoso juego *Candy Crush* reportó tener 356 millones de usuarios únicos, con unas ganancias de 1.300 millones de dólares. La industria de los videojuegos es, así, la segunda con mayor beneficios después de la de libros y supera a la del cine y música. No solo forma parte de nuestro entretenimiento, sino que, cada vez más, se incorpora al mundo de la enseñanza, especialmente para niños.

Por estas razones, los mecanismos de seguridad asociados a los juegos son importantes. Pensados originalmente para combatir la piratería, distribución ilícita de copias de juegos, hoy en día abarcan más campos aparte de estos. Son importantes para evitar modificaciones, realizar acciones no autorizadas durante las partidas en línea y traducciones a idiomas donde la empresa piensa crear mercado.

Este trabajo pretende ofrecer una perspectiva de seguridad sobre este creciente mercado, analizando juegos de las videoconsolas Nintendo DS, Play Station 3 y plataformas móviles como Android y iOS. Se verá cómo se han protegido textos, imágenes y archivos ante modificaciones, con cifrados. Así mismo, se analizarán protocolos de comunicación para servicios en línea, el cifrado en las descargas de archivos, la transmisión segura de código y la protección de contenidos con derechos de autor. En general, estas técnicas, pretenden impedir poder realizar ingeniería inversa sobre el producto final, entendiéndose esta como el proceso de analizar una aplicación para comprender cómo funciona y cómo integra cada uno de sus recursos.

La ingeniería inversa se lleva realizando desde el inicio de la computadoras, en su mayoría con motivos de entretenimiento y enseñanza; ¡aprendiendo cómo funciona un sistema se puede crear uno más robusto y con más características! Existen diversas herramientas para llevarla a cabo, tales como depuradores y desensambladores, aunque son específicas para cada plataforma. En este trabajo se ha utilizado un emulador de código abierto DeSmuME y otro privativo pero *freeware*, No$GBA, para Nintendo DS. Estos se han usado con el objetivo de analizar el código en ensamblador de los juegos y de automatizar tareas de depuración como guardar paquetes de red, exportar datos descifrados de rutinas de código y analizar los archivos cargados. Además, para el desarrollo de este proyecto se han realizado una serie de programas que ayudan al análisis de juegos. También se explican las metodologías seguidas para el desarrollo de los estudios.

Las conclusiones del estudio demuestran una ausencia general en cuanto a protección sobre los archivos con contenidos con *copyright*. Así mismo, se evidencian las técnicas utilizadas por algunas compañías a la hora de ofuscar para evitar modificaciones de archivos, tales como cifrados mediante operaciones `XOR`, algoritmos `HMAC` y de firma digital. En cuanto a los servicios en línea, se han detectado fallos de seguridad a la hora de configurar servidores y fallos en el diseño de los protocolos, que permiten realizar acciones no autorizadas con facilidad. En suma, se deducen unas claras carencias en la provisión de seguridad en los videojuegos, que ponen de manifiesto la necesidad de mejorar los esquemas actualmente dispuestos para ellos.

Data Protection Mechanisms in Video Games

Benito Palacios Sánchez

Keywords: security, video games, reverse engineering, online services, DRM, Nintendo DS.

Abstract

Video games are getting one of the most important elements of our culture. The mobile market boom has made companies behind games like *Candy Crush* very successful. In that specific case, the game has about 356 millions of unique users and 1.300 millions of dollars in benefits. The video game industry is in fact only before the book's one in top of profits and it gets over music and cinema. It is not only about entertainment but also for education, specially for children, for instance, to learn new languages and help with mathematics.

All that benefit makes to think about how to protect against possible 'attacks'. They started fighting piracy with Digital Rights Management techniques, that is, releasing copy of games without permissions from the author, but nowadays it deals with more fields. Mods, cheats in online plays and translations made by fans to languages where the company is going to release a port of the game are only some examples.

This project aims to offer a new perspective about game security. In it, we analyze products from Nintendo DS and Play Station 3 devices and mobile platforms like Android and iOS, to show what kind of techniques has been used to protect both text and images files, as well as sound archives. Communication protocols for online services, download content protection and wireless code transmission will be studied too. The general purpose of all these protection is to avoid doing reverse engineering, that is, disassemble the product to study all its pieces and resources.

Reverse engineering is one of the main topics of this dissertation. Usually it is done as a hobby but, with the rights tools and knowledge it is a powerful way to test the security of a system. Related to video games, it is called *romhacking*, since it is about modify (*hack*) a game (usually distributed in *Read Only Memories*). These project will use common tools like disassemblers and debuggers. For instance, for Nintendo DS games two emulators will be used. The first one, DeSmuME, since it is open source, it will allow to hack its code to automatize task and save network packets. The second one, No$GBA, *freeware*, includes a built-in debugger so it will be used to analyze code and algorithms. Some tools have been developed to help with the task of studying games, like binary searcher, database manager and some game specific exporters. Furthermore, all the methodology followed to figure out formats and algorithms will be explained in detail.

This project concludes with a summary of the mechanism analyzed, with recommendations to improve them. Most copyright contents, that usually contains DRM in virtual stores, are not protected in games. That refers to music, electronic books and videos. However, text and images are usually protected with encryption by using `XOR` operand, `HMAC` algorithm or a digital signature. About the online services, it will show how bad protocol designs has been found, with some vulnerabilities that allow to users to cheat, but also how in some games the download content has a good protection against external modifications and man-in-the-middle attacks.

Índice general

1. Introducción **1**
- 1.1. Motivación . . . 1
- 1.2. Objetivos . . . 1
- 1.3. Organización . . . 2

2. Seguridad en videojuegos **3**
- 2.1. Conceptos de seguridad . . . 4
  - 2.1.1. Cifrado simétrico y asimétrico . . . 4
  - 2.1.2. Algoritmos para integridad . . . 5
  - 2.1.3. Firma digital . . . 5
- 2.2. Seguridad en videoconsolas . . . 5
  - 2.2.1. Nintendo DS . . . 6
  - 2.2.2. Nintendo DSi . . . 6
  - 2.2.3. Nintendo 3DS . . . 6
- 2.3. Ingeniería inversa . . . 7
  - 2.3.1. Legalidad . . . 8

3. Planificación y costes del proyecto **9**
- 3.1. Organización . . . 9
- 3.2. Planificación . . . 10
- 3.3. Presupuesto . . . 10
  - 3.3.1. Recursos . . . 10
  - 3.3.2. Estimación de costes . . . 12

4. Metodología **15**
- 4.1. Análisis de ficheros . . . 15
- 4.2. Depuración de código . . . 17
  - 4.2.1. Búsqueda de archivos en memoria RAM . . . 17
  - 4.2.2. Búsqueda de algoritmos sobre textos . . . 18
  - 4.2.3. Búsqueda de algoritmo sobre imágenes . . . 19
- 4.3. Interceptación de la comunicación . . . 19
- 4.4. Documentación y repositorio . . . 20

5. Traducciones no oficiales **23**
- 5.1. Saga Pokémon en Nintendo DS . . . 24
  - 5.1.1. Pokémon Perla y Diamante . . . 24
  - 5.1.2. Pokémon HeartGold y SoulSilver . . . 26
  - 5.1.3. Pokémon Blanco y Negro . . . 28
  - 5.1.4. Pokémon Conquest . . . 30
- 5.2. Ninokuni: El Mago de las Tinieblas . . . 31
- 5.3. Proyectos abandonados . . . 33

6. Contenido con derechos de autor **35**

6.1. Libros electrónicos . . . 36
6.1.1. Ninokuni: La Ira de la Bruja Blanca . . . 36
6.1.2. 100 Classic Book Collection . . . 37
6.2. Bandas sonoras . . . 38
6.2.1. Osu! Tatakae! Ouendan! . . . 38
6.2.2. Guitar Hero: On Tour . . . 39
6.2.3. Guitar Rock . . . 40
6.2.4. Level-5 . . . 40
6.2.5. Duet . . . 41
6.3. Vídeos . . . 41
6.3.1. Play Station 3 . . . 41
6.3.2. Nintendo DS . . . 42

7. Servicios en línea **45**
7.1. Multijugador . . . 45
7.1.1. Conexión segura en servidores de Nintendo . . . 45
7.1.2. Preguntados . . . 49
7.2. Contenido descargable . . . 49
7.2.1. 100 Classic Books Collection . . . 49
7.2.2. Ninokuni: El Mago de las Tinieblas . . . 50
7.2.3. Duet . . . 50
7.3. Transmisión segura de código . . . 51

8. Resultados y recomendaciones **53**
8.1. Seguridad sobre ficheros . . . 53
8.2. Seguridad en comunicaciones . . . 54

9. Conclusiones **55**
9.1. Trabajo futuro . . . 55

Bibliografía **57**

Acrónimos **59**

Índice de figuras

3.1. Diagrama de Gantt del proyecto. 11

4.1. Contenido de un fichero con imágenes de *Ninokuni*. 15
4.2. Contenido de un fichero comprimido de *Ninokuni*. 16
4.3. Primeros bytes del fichero `PSAR`. 16
4.4. Programas para depurar juegos de Nintendo DS. 17
4.5. Frase con codificación no estándar encontrada por RelativeSearch en *Pokémon*. 19
4.6. Programa de gestión de algoritmos de juegos DataBrithm. 21

5.1. Sistema de ficheros en Pokémon Perla visto con Tinke. 25
5.2. Carpetas y ficheros ofuscados en Pokémon HeartGold. 27
5.3. Comparación de carpetas entre Pokémon Blanco (izq.) y Pokémon HeartGold (der.). . 28
5.4. Imágenes sin cifrar en Pokémon Blanco. 30
5.5. Archivo cifrado (arr.) y descifrado (ab.) de *Ninokuni: El Mago de las Tinieblas*. 32
5.6. Mensaje de error al detectar una partida modificada (izq.) y advertencia (der.) en Ninokuni. 33

6.1. Extracto de texto de un libro de *100 Classic Books Collection*. 37
6.2. Archivos comprimidos en `SDAT` y codificados con IMA-ADPCM visto con Tinke. 38
6.3. *Guitar Grip* necesario para jugar a la saga *Guitar Hero: On Tour*. 39

7.1. Primeros paquetes de una comunicación entre una Nintendo DS y los servidores. . . . 46
7.2. Comunicación entre Nintendo DS y servidor de Nintendo para descargar un fichero. . . 47
7.3. Petición respuesta descifrada entre Nintendo DS y servidor. 48
7.4. Mensajes descifrados del protocolo multijugador de Nintendo DS. En verde el *token*. . 48
7.5. Conexión a los servidores alternativos de Nintendo. 49
7.6. Captura de tráfico con las preguntas y respuestas de una partida de *Preguntados*. . . . 49
7.7. Filas de la base de datos de *Duet* que activan el contenido extra. 50

Índice de tablas

2.1. Tabla de resultados de la operación `XOR`. 5

3.1. Coste de los recursos de personal. 13
3.2. Presupuesto final. 13

5.1. Especificación de tipografías en Pokémon Perla. 25
5.2. Especificación del formato de texto en Pokémon Perla. 26
5.3. Especificación del formato de texto en Pokémon Blanco. 29
5.4. Especificación del formato de texto en Pokémon Conquest. 31

6.1. Especificación de formato `PSAR`. 37
6.2. Especificación de formato `GVD`. 43
6.3. Especificación del formato de *100 Classic Books*. 44
6.4. Especificación del formato de *Guitar Hero: On Tour*. 44
6.5. Especificación del formato `HWAS`. 44

Capítulo 1

Introducción

1.1. Motivación

El primer videojuego se remonta al año 1952 [9], cuando el profesor de informática Alexander S. Douglas creó un juego con gráficos para ordenador. Llamado *Nought and crosses* u *OXO*, se trata de un 'tres en raya' implementado para la computadora *EDSAC* de la Universidad de Cambridge. Este permitía enfrentar a una persona contra la máquina. Seis años más tarde, William Higginbotham desarrolló el juego *Tennis for Two* sobre un osciloscopio, inventando el primer videojuego multijugador. Cuatro años después, un estudiante del *Massachusetts Institute of Technology* (MIT) creó un juego con gráficos vectoriales donde dos naves se enfrentaban, denominado *Spacewar*.

Desde entonces, y 63 años más tarde, existen ocho generaciones de videoconsolas. Las últimas permiten jugar en alta calidad y 60 fps, consiguiendo un alto nivel de detalle y realismo. La industria de los videojuegos ha evolucionado a un ritmo imparable, siendo uno de los sectores que más rápido ha crecido en Estados Unidos [4]. En 2014, se vendieron solo en Estados Unidos 135 millones de juegos, generando unas ganancias de 22 mil millones de dólares. Para 2015, la empresa *Gartner* estima que la venta de videojuegos a nivel global será de en torno a 100 mil millones de euros. En cuanto a España, en 2014 la industria creció un 21 %, facturando 413 millones de euros según la *Asociación Española de Empresas Productoras y Desarrolladoras de Videojuegos y Software de Entretenimiento* [8].

1.2. Objetivos

En el contexto anteriormente planteado, el objetivo que persigue este trabajo es analizar tres diferentes aspectos de seguridad sobre videojuegos. En cada uno se ofrecerán varios análisis para mostrar vulnerabilidades y puntos fuertes de cada videojuego y, ofrecer recomendaciones para mejorar su seguridad.

Para ello se habrá de determinar el ámbito de las problemáticas, identificando qué tipo de contenido deben ser protegido y por qué. Sobre esto, habrá de realizarse un análisis del estado actual y en qué medida afecta a la industria.

Se han escogido juegos de la Nintendo DS para la mayoría de los casos, por la documentación que existe sobre el hardware de la consola y emuladores con capacidades de depuración. Los resultados del análisis de estos juegos son el objetivo de la memoria, mostrando qué técnicas de protección de datos se han implementado y cómo se podría haber mejorado.

Este estudio surge del reciente interés por la edición de videojuegos al que se llama *romhacking*. El nombre proviene del inglés *modificación no autorizada* (*hack*) sobre un archivo que originalmente viene en dispositivos de solo lectura (*Read Only Memory, ROM*). Estas modificaciones se realizan aplicando conceptos de ingeniería inversa sobre los archivos, es decir, a partir del producto final se extraen sus recursos y se averigua cómo funciona. Existen comunidades en Internet dedicadas a este propósito[1], donde se comparte información y utilidades. Los tres tipos de modificaciones y estudios más frecuentes en este movimiento, y sobre los que se basará este trabajo, son:

1. Las traducciones no oficiales son llevadas a cabo por personas externas a una empresa, de manera que, editan los archivos del juego para ofrecer un parche que traduce el juego a un idioma. Relacionado con este tema están los *mods*, modificaciones sobre un juego para ofrecer una versión alternativa.
2. Los contenidos con derechos de autor será el otro tema a tratar, y en ello en relación a los videojuegos con contenido sobre el que típicamente se le aplican algoritmos de protección como libros electrónicos, música y vídeos.
3. El tercer aspecto será los servicios en línea, tales como los protocolos de comunicación entre consola y servidor y la seguridad sobre los ficheros transmitidos.

1.3. Organización

La memoria se ha organizado de forma que primero se presentará la metodología a seguir en el análisis de los videojuegos, a continuación los resultados de los análisis y finalmente una conclusión.

En el Capítulo 2 se introducirá la edición de videojuegos, la seguridad que habitualmente se implementa y aspectos legales para realizar ingeniería inversa.

En el Capítulo 3 se mostrarán los apartados y tareas en los que se ha dividido y planificado el trabajo, además de los recursos y los costes estimados.

El Capítulo 4 explica las diferentes metodologías empleadas a la hora de analizar un aspecto de un videojuego. También se hará referencia a los programas desarrollados durante el trabajo para facilitar los análisis y se detallará su uso y funcionamiento.

El Capítulo 5 comenta los resultados obtenidos tras analizar juegos que han sido objeto de traducciones no oficiales. La investigación se concentró en averiguar los algoritmos empleados para ofuscar textos e imágenes. Incluye también un estudio sobre las causas de abandono de estos proyectos.

En el Capítulo 6 se habla sobre los contenidos con derecho de autor. Para ello, se han escogido una serie de juegos que incluyen libros electrónicos, canciones o vídeos; y se han analizado los posibles algoritmos para proteger dichos contenidos.

El Capítulo 7 muestra los resultados de analizar diferentes servicios en línea. Se explicarán los mecanismos de seguridad de los servidores de Nintendo para Nintendo DS, comentando la metodología seguida y los fallos de seguridad descubiertos. También se comentará la seguridad implementada sobre videojuegos para jugar en multijugador y descargar contenidos extras. El capítulo termina con un análisis sobre cómo se transmite código ejecutable en la Nintendo DS de forma segura entre consolas.

El Capítulo 8 resume los algoritmos encontrados así como sus puntos débiles y fuertes y se proponen algunas técnicas para reforzar la seguridad global en videojuegos.

La memoria concluye con el Capítulo 9, donde se detallan los conocimientos aprendidos y habilidades desarrolladas con este trabajo. Incluye una sección con líneas de trabajo sobre los que se podría continuar avanzando en el trabajo aquí realizado.

[1] `http://romhacking.net/`

Capítulo 2

Seguridad en videojuegos

Este trabajo explora los conceptos de seguridad y videojuegos. Unos términos que en principio no se suelen relacionar, a no ser que se hable sobre la piratería, fenómeno que aumenta cada día más, esperando un crecimiento del 22 % para 2015 [3]. Esta fuente menciona también la ingeniería inversa como '*usando herramientas genéricas, los hackers pueden convertir rápidamente binarios desprotegidos en código fuente, volver a empaquetarlos y distribuirlos*'.

Comúnmente se asocia el término de *hacker* a una persona que maliciosamente investiga un programa. Es una mala interpretación dada por medios y películas, ya que realmente se habría de hablar de *cracker*. El nombre de *hacker* nació en 1961, en los laboratorios del MIT, para denominar a los estudiantes que dominaban con destreza la programación. A día de hoy, según el RFC 1392[1] se define *hacker* como:

«*Hacker: A person who delights in having an intimate understanding of the internal workings of a system, computers and computer networks in particular. The term is often misused in a pejorative context, where 'cracker' would be the correct term.*»

«Hacker: Persona apasionada por entender cómo funciona internamente y en detalle, un conjunto de sistemas, ordenadores y redes de ordenadores. Generalmente se usa de forma incorrecta en un contexto peyorativo, debiendo usarse de modo más correcto 'cracker'.»

De esta forma, este mismo RFC define *cracker* como:

«*Cracker: A cracker is an individual who attempts to access computer systems without authorization. These individuals are often malicious, as opposed to hackers, and have many means at their disposal for breaking into a system.*»

«Cracker: Individuo que intenta acceder a un sistema de ordenadores sin autorización. Estos individuos son generalmente maliciosos, en oposición a los 'hackers', y tienen intereses ocultos en su intento por romper el sistema.»

Este trabajo muestra la seguridad de los juegos con el único propósito de enseñanza, como menciona Andew Huang [13, p. 7]: *For every copyright protection scheme that is defeated by a hacker, there is someone who learned an important lesson about how to make a better protection scheme.* (*Para cada esquema de protección con copyright que un hacker rompe, hay alguien que aprende una importante lección sobre cómo hacer un esquema de protección más robusto.*).

[1] `https://tools.ietf.org/html/rfc1392`

2.1. Conceptos de seguridad

El concepto de seguridad se puede definir en tres puntos: la condición de un sistema como resultado del establecimiento y mantenimiento de medidas de protección; la condición de un sistema en el cual sus recursos están libres de accesos no autorizados y de cambios accidentales no autorizados, destrucción o, pérdida; medidas tomadas para proteger un sistema.[2]. Esta puede estar referida a tres ámbitos:

- La seguridad de la información trata sobre la protección de datos guardados.
- La seguridad en las comunicaciones se refiere a la protección en la transmisión de datos
- La seguridad de sistemas se centra en la protección de entidades por los que pasa la información.

Este trabajo abarca los dos primeros puntos, habiendo una pequeña introducción sobre el tercero en el Apartado 2.2 de seguridad en videconsolas.

Max Kilger, investigador de ciberseguridad, resume la motivación de las amenazas con las siglas `MEECES` del inglés dinero, ego, entretenimiento, ideología, entrada en grupos sociales y estatus social.

Con el objetivo de hacer frente a estas amenazas, existen varios modelos de seguridad con los aspectos que debe cumplir un sistema considerarse protegido. El más conocido se denomina CIA, por las iniciales de los tres principales puntos recogidos a continuación. A este modelo se añaden cuatro puntos más, resultando la siguiente lista:

- Confidencialidad: la información solo puede estar accesible por los entes autorizados.
- Integridad: se ha de garantizar que la información no ha sido modificada de manera no autorizada.
- Disponibilidad (*Availability*): la información ha de estar accesible cuando se solicita.
- Autenticación: se ha de garantizar quién es el otro extremo.
- Disuasión: se ha de evitar cualquier motivación para realizar ataques.
- No repudio: se ha de garantizar que la operación se realizó.
- Recuperación: se puede devolver el sistema su estado inicial tras un ataque.

2.1.1. Cifrado simétrico y asimétrico

La técnica de cifrado se emplea para garantizar la confidencialidad. En líneas generales se basa en aplicar un algoritmo con una clave sobre un bloque de datos, denominado texto plano. El resultado es otra representación del texto plano de manera que no se puede recuperar la información original sin conocer los detalles del cifrado.

Estos algoritmos se basan en operaciones de sustitución y transposición, reordenación de bloques de datos. Se clasifican en dos tipos principalmente: simétricos y asimétricos.

El cifrado simétrico hace referencia a la existencia de una misma clave para cifrar y descifrar. Esta clave la tienen que conocer los dos extremos para poder establecer una comunicación con éxito. Algunos de los algoritmos más conocidos son `IDEA`, `3DES`, `AES` y `RC4`.

Sin embargo, no hace falta aplicar operaciones tan complejas cuando solo se desea ofuscar el texto, como sucede en los videojuegos. En estos casos, la operación `XOR` sirve a este propósito (Tabla 2.1). El primer operando sería el byte a cifrar y el segundo un valor de clave. Esta operación tiene el problema de que si se aplica sobre un byte con valor cero, el resultado será el segundo operando, es decir, la clave. Otra desventaja es que, conociendo una sección del texto plano y teniendo el texto cifrado, el resultado de aplicar `XOR` entre ambos sería la clave. Para evitar estos dos problemas, se usa una clave que cambia tras cada ejecución, de forma que mediante un conjunto de bytes nulos solo se recuperaría un estado temporal de la misma.

[2] `https://tools.ietf.org/html/rfc4949`

Tabla 2.1: Tabla de resultados de la operación `XOR`.

Primer operando	Segundo operando	Resultado
0	0	0
0	1	1
1	0	1
1	1	0

El cifrado asimétrico, a pesar de ser más lento que el simétrico, soluciona el problema de compartir la misma clave usada para cifrar y descifrar. De esta forma, cada extremo posee una clave y no necesita conocer la otra. Un mensaje cifrado con la primera clave solo podría ser descifrado con la segunda, y viceversa. Este tipo de algoritmo se usa en las comunicaciones a través de redes inseguras como Internet. Cuando se genera el par de claves, una se suele mantener secreta (clave privada) y la otra se comparte entre varias entidades (clave pública). El algoritmo más utilizado es `RSA`, que soporta tanto cifrado y descifrado como firma digital e intercambio de claves.

2.1.2. Algoritmos para integridad

Los algoritmos para integridad tienen como objetivo asegurar que el mensaje no ha sido modificado. Para ello, mediante una serie de operaciones, se ofrece un resumen de tamaño fijo de los datos de entrada. Los algoritmos deben cumplir que las colisiones (coincidencias de resumen entre dos bloques de datos distintos) sean no reproducibles. Estos se clasifican por el número de bits del resultado, siendo los más frecuentes `MD5`[3], con 128 bits, y `SHA-1`, con 160 bits.

2.1.3. Firma digital

La firma digital es una técnica para proveer tanto autenticidad como integridad sobre un mensaje. Consiste en aplicar un algoritmo de integridad sobre un bloque de datos y, el resultado cifrarlo con la clave privada del emisor. De esta forma, solo la clave pública correspondiente a esta entidad podrá descifrarlo, asegurando que ha sido esta quien ha realizado la operación. Además, al tener el resumen se puede comprobar que el mensaje no ha sido modificado. Lo frecuente es emplear `SHA-1` para la integridad y `RSA` para cifrar.

2.2. Seguridad en videoconsolas

La seguridad cada vez más se confía en la consola, en lugar de sobre el propio juego. El principal problema que esto genera es que una vez que esta protección se consigue romper, todos los juegos quedan expuestos.

A principios de la era de los videojuegos, la piratería era poco común, principalmente por la dificultad en encontrar el hardware necesario para romper los sistemas de protección. Los juegos se distribuían en cartuchos de solo lectura, ya que no existían memorias se uso genérico como SD, ni protocolos de comunicación como USB.

Con la introducción de la Nintendo DS, esto cambió. Los juegos se distribuyen en pequeños cartuchos de tamaño similar a una tarjeta SD. Debido a la existencia de tarjetas MicroSD, se pudieron crear dispositivos que simulan ser un juego comercial y que permiten ejecutar código no autorizado por Nintendo. Estos se conocen como *flashcards* y se basan en *exploits* que consiguen saltarse las limitaciones del sistema de la consola para, simulando ser un juego, comenzar la ejecución de otro.

[3]Este algoritmo es inseguro.

2.2.1. Nintendo DS

En el caso de la Nintendo DS, los juegos incorporan una lógica para transmitir, mediante un protocolo, datos cifrados a la consola [15]. Este cifrado se basa en dos claves, la primera constante y la segunda generada en cada ejecución. Se cifran los comandos enviados por la consola al cartucho y la respuesta con los datos del juego. Aparte del cifrado, la *BIOS* y *firmware* realizan una comprobación sobre la cabecera del juego. En concreto, hay una región donde se encuentra el logo de Nintendo ofuscado [4]. El propósito es que, al contener datos con derechos de autor como es el logo de la compañía, los juegos no se podrían distribuir sin la autorización de Nintendo. Un caso similar fue llevado a juicio en Estados Unidos, perdiendo Sony, la empresa que pretendía evitar estas actuaciones [7, p. 18].

El caso descrito sucedió con la consola Sega Genesis, cuando la empresa Accolade, en lugar de afiliarse con Sega para desarrollar juegos, decidió realizar ingeniería inversa y crear sus productos en base a esa información. Esta compañía determinó que la palabra 'SEGA' tenía que estar contenida en la cabecera del juego para hacerlo funcionar. Sin embargo, esos caracteres están protegidos con *copyright* por Sega y en base a esto, llevó a cabo una demanda. Finalmente, la corte le dio la razón a Accolade ya que no había copiado código de Sega y beneficiaba al mercado introduciendo competencia.

2.2.2. Nintendo DSi

Un sistema más robusto se introdujo con la Nintendo DSi. El formato de los juegos se mantiene pero, en aquellos juegos exclusivos para la nueva generación, se añade una firma digital usando la clave privada de Nintendo. El sistema operativo de la consola comprueba la firma y, de ser inválida, el juego no se ejecuta.

Mediante este procedimiento, las *flaschard* dejaron de funcionar. No se podía ni generar una firma digital válida, ni utilizar una existente porque, al modificar el código del juego, la firma sería inválida.

El agujero de seguridad vino junto a las malas implementaciones de algunos juegos. Modificando los archivos de guardado de ciertos juegos, se conseguía provocar un fallo del juego (*buffer overflow*), de forma que el siguiente código que ejecutaba era el almacenado en el propio archivo de guardado. Esto implica que distribuyendo un juego comercial con este fallo junto a una partida preparada para explotarlo, se podían crear *flashcard* que ejecutasen cualquier código contenido en el archivo de la partida.

2.2.3. Nintendo 3DS

La siguiente generación de consolas de Nintendo aumentó más la seguridad. Los juegos distribuidos vienen protegidos con un cifrado simétrico implementado sobre un módulo hardware de la consola. Cuando se piden datos al cartucho, estos pasan por el módulo de descifrado de la consola y se almacenan en la memoria `RAM`. Se trata de un módulo diseñado específicamente para la consola, encontrándose la clave sobre las pistas del chip, por lo que no se puede averiguar.

Para incrementar la seguridad se colocaron los componentes de la consola estratégicamente, de forma que era complicado extraer y acceder a la memoria `RAM`. A pesar de ello, hubo personas que consiguieron acceder, pudiendo leer los datos descifrados del juego e incluso alterar las instrucciones almacenadas en la memoria para ejecutar código que forzara descifrar del juego completo. Fue este, así, un proceso manual que permitió encontrar una serie de *exploits*[5] que se aprovechaban de nuevo de fallos de seguridad en los archivos de guardado para ejecutar código descifrado directamente.

[4] `http://pleonet.blogspot.com.es/2013/08/logo-de-nintendo-en-gba-y-nds.html`
[5] `http://smealum.net/ninjhax/`

2.3. Ingeniería inversa

«***Reverse engineering*** *is the process of analyzing a subject system to identify the system's components and their inter-relationships, and to create representations of the system in another form or at higher levels of abstraction.*»

«**La ingeniería inversa** es el proceso de analizar un sistema para identificar sus componentes y relaciones y, crear una representación del sistema en otro formato o a un nivel más alto de abstracción.»

Con estas palabras definió el compendio del proyecto europeo REDO en 1993 el término *ingeniería inversa* [6, p. 17]. Se trata de un proyecto enmarcado dentro del plan *European Strategic Program on Research in Information Technology* (ESPRIT). El trabajo realizado por 11 organizaciones de 7 países tenía como objetivo tratar el problema del mantenimiento de software mediante técnicas de ingeniería inversa. Para ello se propusieron los siguientes objetivos:

1. Validar el software existente.
2. Acoplar el software y su documentación.
3. Usar métodos formales en mantenimiento de software.
4. Mejorar la usabilidad del software existente mediante mejores interfaces de usuario.
5. Desarrollo y mejora de herramientas para reestructurar el código fuente y el control de trabajo.
6. Desarrollo de un lenguaje genérico con el que se pueda expresar la semántica de diferentes lenguajes de programación y control de trabajo.

Este proyecto identificó dos tareas principales a la hora de realizar un trabajo de ingeniería inversa [6, p. 17]:

- *Redocumentación*. Proceso por el que se crea una representación semánticamente equivalente con el mismo nivel de abstracción. En este trabajo se ha llevado a cabo mediante el desarrollo de ciertos programas que procesan ficheros de igual forma que lo hace un videojuego.
- *Recuperación de diseño*. Proceso que involucra identificar el diseño en niveles más altos de abstracción que aquellos se que pudieran ver, examinando el sistema en sí. Esta tarea se ha llevado aquí documentando los algoritmos estudiados.

En proceso de desarrollo común es el de programar una aplicación en un lenguaje de alto nivel y, mediante un software denominado *compilador*, convertir el texto escrito en una serie de bytes que el ordenador es capaz de interpretar para ejecutar operaciones a nivel de hardware. La tarea de depuración, en el caso de la ingeniería inversa, trata sobre analizar esos bytes, instrucciones que el ordenador interpreta, y a partir de ellas entender el diseño original en alto nivel. Existen programas llamados *descompiladores* que automatizan esta tarea para diferentes lenguajes de programación, como por ejemplo, el que incluye *IDA Pro* para convertir lenguaje ensamblador en lenguaje C.

La forma de evitar este proceso, aparte de aplicar cifrados a nivel de hardware como hace la Nintendo 3DS (Apartado 2.2.3), es mediante ofuscación de código. Este proceso consiste en transformar el código de forma que es menos legible, pero mantiene la funcionalidad [7, p. 344]. Debe de cumplir además que las transformaciones aplicadas no sean sencillas de revertir, de forma que no se pudiera crear un *desofuscador* y que el sobrecoste que este proceso conlleva no afecte al rendimiento. El nivel de complejidad que añade esta técnica se llama *potencia*, y se puede medir con software convencional que toma en cuenta factores como el número de funciones y la profundidad de anidado que tiene una secuencia de código.

Un sistema podría considerarse seguro una vez que el esfuerzo y tiempo que requiere romper su seguridad es mayor al valor del producto y su validez. Un ejemplo sería que el coste de distribuir copias no legales de un juego sea mayor al coste del mismo original, o que el tiempo empleado para averiguar una clave sea mayor a la frecuencia con la que el sistema la cambia.

En cuanto a la ingeniería inversa en videojuegos, no es distinta a los procedimientos seguidos sobre aplicaciones. Esta se centra más sobre los recursos y sus formatos que en el código y funcionalidad[6].

2.3.1. Legalidad

La ingeniería inversa no está exenta de controversia a la hora de determinar su legalidad. El punto 14 de la *directiva 2009/24/CE sobre la protección jurídica de programas de ordenador*[7] dice:

«No debe impedirse a la persona facultada para utilizar el programa de ordenador que realice los actos necesarios para observar, estudiar o verificar su funcionamiento, siempre que dichos actos no supongan infracción de los derechos del autor sobre el programa.»

Es decir, siempre que se posean los derechos de acceder a un contenido, no se debe prohibir el derecho a estudiar este. Sin embargo, las directivas europeas son recomendaciones que los países miembros implementan con sus propias leyes. En el caso de España, se recoge en número 97 del BOE[8] en el título VII, *Programas de ordenador*, artículo 100, *límites a los derechos de explotación*:

«El usuario legítimo de la copia de un programa estará facultado para observar, estudiar o verificar su funcionamiento, sin autorización previa del titular, con el fin de determinar las ideas y principios implícitos en cualquier elemento del programa, siempre que lo haga durante cualquiera de las operaciones de carga, visualización, ejecución, transmisión o almacenamiento del programa que tiene derecho a hacer.»

donde se recoge el mismo contenido que la directiva europea recomienda. Esto, además, está respaldado por la resolución de un juicio en la Corte Suprema de la Unión Europa en esta materia[9]. En ella, la compañía *World Programming Ltd* fue demandada por *SaaS Institute Inc.* al desarrollar un software a partir de conocimientos adquiridos mediante ingeniería inversa, que replicaba el funcionamiento del programa de la compañía demandante. La corte señaló que '*There is no copyright infringement [when a software company without access to a program's source code] studied, observed and tested that program in order to reproduce its functionality in a second program. (No se infringe el copyright [cuando una compañía de software sin acceso al código fuente del programa] estudia, observa y prueba el problema para reproducir su funcionalidad en un segundo programa)*'.

En el caso de Estados Unidos, la ley *Digital Millennium Copyright Act* (DMCA) lo prohíbe a no ser que se cuente con el permiso del autor. El Capítulo 12 del libro *Hacking the Xbox*, escrito por Lee Tien, abogado de la *Electronic Frontier Foundation* (EFF), contiene un completo análisis al respecto. Incluye una cita de la Corte Suprema americana, refiriéndose a la ingeniería inversa como *una parte esencial de la innovación* [13, p. 180].

[6] En la siguiente dirección se encuentra una guía de introducción a la ingeniería inversa para Nintendo DS: `http://gbatemp.net/threads/gbatemp-rom-hacking-documentation-project-new-2014-edition-out.73394/`

[7] `http://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:32009L0024&from=EN`

[8] `http://boe.es/buscar/act.php?id=BOE-A-1996-8930&tn=1&p=19980307&vd=#a100`

[9] `http://www.bloomberg.com/news/articles/2012-05-02/copyright-can-t-block-software-reverse-engineering-court`

Capítulo 3

Planificación y costes del proyecto

Este capítulo explica cómo se ha estructurado y planificado el trabajo, así como los costes que ha supuesto. Analizada la problemática y su contexto, se ofrecerá un enfoque global en cuanto a diferentes aspectos sobre videojuegos pocos tratados en la literatura.

3.1. Organización

El proyecto se ha estructurado en cinco tareas principales explicadas a continuación:

1. Caracterización de algoritmos. El objetivo es introducirse en el tema, estudiando los mecanismos actuales para proteger datos en videojuegos. También se estudiará qué problemáticas que se analizarán en profundidad durante el trabajo. Están implicadas las siguientes labores:
   - Buscar información sobre seguridad en videojuegos.
   - Buscar información sobre los algoritmos de *Digital Resource Management* (DRM) actuales.
   - Identificar categorías sobre las que los algoritmos se centran.
   - Identificar las características de cada categoría.
2. Escoger juegos para el estudio. Una vez analizada la situación actual, e identificados los algoritmos deseados para estudiar, el siguiente bloque estudiará qué juegos serán objeto del trabajo. Así, los puntos a desarrollar son:
   - Buscar información sobre proyectos de edición abandonados.
   - Identificar juegos con contenidos con derechos de autor.
   - Priorizar juegos por aquellos más vendidos.
3. Desarrollo de software de ayuda. Antes de comenzar la investigación se planificó un periodo de tiempo para el desarrollo de utilidades para su posterior uso. Se incluyó el estudio de un posible desarrollo de una utilidad opcional que, si bien existían soluciones que podían usarse para el trabajo, no eran de código abierto. Las subtareas, así, a desarrollar aquí son:
   - Crear *script* para clasificar juegos por número de ventas.
   - Crear un gestor para una base de datos sobre algoritmos.
   - Crear programas de búsqueda y modificación binaria avanzada.
   - (Opcional) Desarrollar depurador de código remoto para *Nintendo DS* (NDS). Se optó finalmente por usar el emulador *freeware No$GBA*.
4. Análisis de juegos. Creados los programas de ayuda, se comenzó con el análisis de los juegos. Se dividieron en tres rondas para ir agrupando los resultados obtenidos:

- Reunir y complementar la información sobre juegos previamente estudiados.
- Primera ronda de siete juegos.
- Segunda ronda de siete juegos.
- Tercera ronda de siete juegos.
- Analizar servicios en línea.

5. Redacción de memoria. Esta tarea se refiere al desarrollo de un documento que explique todos los resultados del trabajo. Para ello, se prevén las siguientes actuaciones:
   - Aprender LaTeX.
   - Crear una plantilla para el trabajo.
   - Desarrollo de la memoria.

3.2. Planificación

El trabajo se ha planificado para realizar las cinco tareas descritas anteriormente, en el plazo de ocho meses descontando festivos. Comenzando en octubre, el análisis del problema planteado y el estudio del arte ocuparán los dos primeros meses de trabajo. Juntando el desarrollo de software de ayuda, ambas partes se estimaron que tomarían hasta principios de 2015.

Desde comienzos de año hasta Semana Santa se planificó la realización de los análisis de los juegos. La estimación fue realizar un análisis de siete juegos cada mes y medio. Una vez terminado el análisis, se revisaron todos los resultados y se redactaron una serie de conclusiones que tomarían hasta el mes de mayo.

El último bloque del trabajo sería el desarrollo de la memoria, incluyendo el aprendizaje del entorno LaTeX. Esto se estimó que duraría un mes y medio hasta finales de junio. En la Figura 3.1 se muestra un diagrama de Gantt a página completa con las temporización global del proyecto.

3.3. Presupuesto

La realización de este trabajo ha supuesto una serie de costes. A continuación se detallan tanto los recursos empleados, como la estimación del coste de los mismos.

3.3.1. Recursos

En la elaboración de este trabajo se han necesitado tanto recursos de personal como materiales. Estos se detallan a continuación.

Recursos de personal

Las siguientes personas han participado en el desarrollo del trabajo.

- D. Benito Palacios Sánchez, alumno del Grado en Ingeniería de Tecnologías de Telecomunicación en la Universidad de Granada, en calidad de autor del trabajo.
- Profesor Dr. D. Pedro García Teodoro, Catedrático de la Universidad de Granada adscrito al área de Ingeniería Telemática en el Departamento de Teoría de la Señal, Telemática y Comunicaciones, en calidad de tutor del proyecto y supervisor de la propuesta.

Recursos materiales

Los recursos materiales se han dividido en dos secciones detalladas a continuación.

	Tareas	Fechas	Oct 40	42	Nov 44	46	48	Dic 50	52	Enero 02	04	Feb 06	08	Marzo 10	12	14	Abril 16	18	Mayo 20	22	Junio 24	27
Caracterización de algoritmos	Situación actual DRM	29/09 – 01/10																				
	Algoritmos existentes DRM	02/10 – 09/10																				
	Identificar categorías de algoritmos	10/10 – 12/10																				
	Identificar las características de algoritmos	13/10 – 19/10																				
Escoger juegos a estudiar	Encontrar proyectos abandonados	20/10 – 26/10																				
	Encontrar juegos con contenido con *copyright*	27/10 – 05/11																				
	Ordenar juegos por ventas	06/11 – 09/11																				
Desarrollo de software de ayuda	Clasificador de juegos	27/10 – 09/11																				
	Depurador de código a distancia (GDB Stub)	10/11 – 23/11																				
	Gestor de base de datos de resultados	24/11 – 07/12																				
	Buscador y editor binario avanzado	08/12 – 21/12																				
Análisis de juegos	Juegos ya estudiados	08/12 – 04/01																				
	Primera ronda (7 juegos)	22/12 – 15/02																				
	Segunda ronda (7 juegos)	06/02 – 29/03																				
	Tercera ronda (7 juegos)	16/03 – 10/05																				
	Analizar servicios en línea	27/04 – 24/05																				
Memoria	Redactar	25/05 – 05/07																				

Figura 3.1: Diagrama de Gantt del proyecto.

Hardware

- Ordenador portátil Asus X66IC.
- Teléfono móvil iOS con *jailbreak* iPhone 4.
- *Router Linksys* modelo WRT54G.
- Cable *Ethernet* de 2 metros.
- 18 juegos para Nintendo DS, 2 para iOS y 1 para Play Station 3.

Software

- Sistemas operativos Fedora 20 y 22.
- Gestor de control de versiones *git*.
- Editor de texto *Atom*.
- Entorno de desarrollo integrado *MonoDevelop*.
- Compilador para C# *mono*.
- Intérprete de python.
- Emuladores para Nintendo DS con capacidades de depuración *DeSmuME* y *No$GBA*.
- Distribución de LaTeX *texlive*.
- Editor de imágenes *GIMP*.
- Analizador de capturas de tráfico *Wireshark*.
- Visor de bases de datos *sqliteman*.
- Explorador de archivos de juegos para Nintendo DS *Tinke*.
- Editor hexadecimal *wxHexEditor* y *HxD*.

3.3.2. Estimación de costes

La estimación de costes se realiza en base a los recursos utilizados y comentados en el apartado anterior, así como la temporización detallada en la Sección 3.2.

Coste de personal

La duración del trabajo se ha estimado en unas 378 horas siguiendo el siguiente desglose:

- Análisis del estado del arte y planificación: 20 horas.
- Desarrollo del software de ayuda: un mes a media jornada de lunes a viernes, 100 horas.
- Análisis de juegos: 8 horas de media por juego y 21 juegos, 168 horas.
- Redacción de la memoria: medio mes a media jornada de lunes a viernes y jornada completa los fines de semana, 90 horas.

Las reuniones con el tutor eran cada dos semanas con una duración media de 30 minutos. Comenzando desde septiembre hasta julio y descontando festivos se estiman unas 20 reuniones, resultando en 10 horas. Además se han de contar las diferentes discusiones previas al trabajo y las revisiones de la memoria, sumando un total de 17 horas.

Tabla 3.1: Coste de los recursos de personal.

Concepto	Coste	Cantidad	Total
Desarrollo	20 euros/h	378 h	7.560 euros
Supervisión	100 euros/h	17 h	1.700 euros
			Total: 9.260 euros

Basándose en publicaciones sobre el salario de un recién graduado en Ingeniería de Tecnologías de Telecomunicación, se estima un sueldo medio de 20 euros por hora de trabajo. En cuanto a la consultaría del tutor, teniendo en cuenta la categoría de catedrático de la Universidad de Granada, se estima un sueldo medio neto de 100 euros por hora de trabajo.

El coste total de este apartado se muestra en la Tabla 3.1, ascendiendo a 9.260 euros, NUEVE MIL DOSCIENTOS SESENTA EUROS.

Coste de materiales

Los materiales, como se comentaron en el apartado de recursos, se dividen en hardware y software.

El coste del hardware usado se ha estimado teniendo en cuenta su vida útil y precio inicial. El ordenador portátil valorado en 600 euros, con una vida útil de 7 años, ha supuesto un coste total de 65 euros para la realización del trabajo en 9 meses. Tanto el teléfono móvil, el *router* y el cable *Ethernet* han sido utilizados durante periodos de tiempo muy cortos y tienen una vida útil muy larga, por lo que su precio no se incluye en el cálculo total.

Para el estudio de los juegos, se tuvo que adquirir una copia física legal de los mismos. Los juegos para Nintendo DS, al ser no ser una consola de última generación, se pueden adquirir por una media de 15 euros por copia. El juego para Play Station 3 utilizado se puede adquirir en la tienda virtual de *Sony* por 5 euros. Finalmente, los dos juegos para plataformas móviles se encuentran de forma gratuito durante periodos de oferta. Esto hace un total de `18 juegos * 15 euros/juego + 1 juego * 5 euros/juego = 275 euros`.

Refiriéndose al primero, se escogieron dando prioridad a soluciones de software libre para poder estudiarlos y modificarlos. En caso de no existir se analizaron por precio, de manera que en cuanto a este apartado no hubo costes.

Presupuesto final

El presupuesto total se basa en los cálculos descritos en la secciones anteriores. Además, se añade un suplemento por costes indirectos tales como manutención, electricidad y conexión a Internet, de un total de 200 euros para todos los meses de trabajo.

El cálculo de este presupuesto se muestra en la Tabla 3.2, haciendo un total de 9.800 euros, NUEVE MIL OCHOCIENTOS EUROS.

Tabla 3.2: Presupuesto final.

Concepto	Coste
Recursos de personal	9.260 euros
Recursos de material	340 euros
Gastos indirectos	200 euros
Total:	**9.800 euros**

Capítulo 4

Metodología

Este capítulo se centrará en explicar las diferentes metodologías, técnicas y programas que se han usado durante el desarrollo del proyecto. No existe un único método cuando se realiza un trabajo de ingeniería inversa ya que cada juego es distinto, con formatos diferentes. Debido a esto, en lugar de proporcionar unos pasos exactos, se detallarán los razonamientos seguidos a la hora de estudiar ficheros y analizar código.

4.1. Análisis de ficheros

Una vez decidido el juego a analizar, se debe reunir información como es el desarrollador, el año de lanzamiento y género del juego. Esta proporciona indicios sobre el tipo y versión del formato de los ficheros que contiene. A continuación se pueden usar programas de exploración de juegos como Tinke[1], para reconocer los tipos de formato estándar, analizar el contenido de las carpetas y exportar los archivos.

El interés, sin embargo, es analizar aquellos formatos nuevos que codifican recursos como imágenes y textos. Los nombres y directorios de los archivos son la primera referencia sobre el tipo de contenido. Por ejemplo, tomando el juego de *Ninokuni* para Nintendo DS como referencia, el archivo `/data/UI/Menu/Skin/2/CheckSheet/bg_a.n2d` debe contener elementos de la interfaz gráfica (*GUI*) del menú (*Menu*) del juego. Además, la abreviatura *bg* se utiliza para describir imágenes de fondo de pantalla (*background*). Mirando el contenido de este archivo con un visor hexadecimal, se puede ver que en la posición `0x54` se encuentran los caracteres `RLCN`, correspondientes a la cabecera de un formato estándar en la Nintendo DS (Figura 4.1).

[1] `https://github.com/pleonex/tinke`

Offset	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0123456789ABCDEF
000000h	4E	50	43	4B	54	00	00	00	09	00	00	00	54	00	00	00	NPCKT ○ T
000010h	28	02	00	00	7C	02	00	00	70	B5	00	00	00	00	00	00	(☻ |☻ p╡
000020h	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000030h	00	00	00	00	00	00	00	00	00	00	00	00	EC	B7	00	00	∞╖
000040h	24	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	$♠
000050h	00	00	00	00	52	4C	43	4E	FF	FE	00	01	28	02	00	00	RLCN ■ ☺(☻

Figura 4.1: Contenido de un fichero con imágenes de *Ninokuni*.

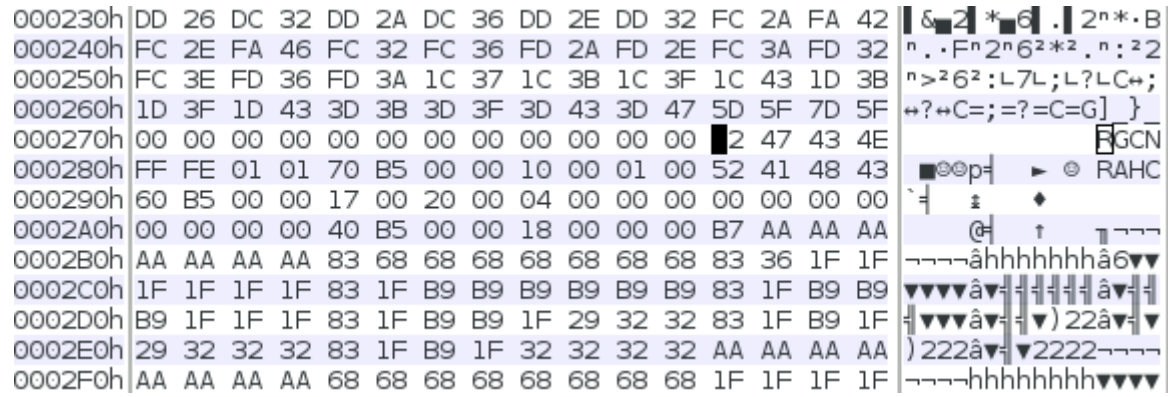


Figura 4.2: Contenido de un fichero comprimido de *Ninokuni*.

Encontrar esta cabecera indica que este archivo contiene varios ficheros, en un formato similar a `zip`, ya que de otro modo estaría al comienzo de los datos. El análisis se centra en saber cómo extraer los ficheros para luego determinar qué datos tienen cada uno de ellos. Se parte sabiendo que la posición del primer fichero es `0x54`, por lo que se buscará este valor en la cabecera del formato. En la posición `0x0C` se observa ese valor, seguido de `0x228`. Frecuentemente, después de la posición de un fichero se indica su tamaño, por lo que habrá que comprobar si el primer fichero termina en la posición `0x54 + 0x228 = 0x27C`. En la Figura 4.2 se ve que en esa posición aparece la cabecera `RGCN`, estándar para otro formato de archivo, por lo que es el comienzo de otro fichero.

Esto corrobora la estructura que se intuía, por cada fichero hay 8 bytes indicando posición y tamaño. Para determinar el número de ficheros se puede calcular cuántos archivos especifica esa *tabla de contenidos*, restando el principio del primer fichero a la posición de la primera entrada y dividiendo por el número de bytes dedicado a cada entrada: `(0x54 - 0x0C) / 0x08 = 9`. El resultado es que se han especificado 9 ficheros, valor que coincide con el encontrado en la posición `0x08` (Figura 4.1).

Esta forma de razonar es la que se ha empleado para averiguar acerca de los formatos estudiados en el trabajo. Falta averiguar el contenido de cada fichero. Estudiando los formatos más comunes, se puede identificar (Figura 4.2) que al principio hay colores, pues cada valor de 16 bits está próximo al siguiente. Al final, hay información sobre píxeles, en concreto el índice al color de la paleta, ya que se repiten muchos valores que están próximos. Esto concuerda con el hecho de que un píxel en una imagen suele tener a su alrededor píxeles de color similar.

Otro ejemplo se encuentra en los archivos de tipo `PSAR` (Figura 4.3) analizados en la Sección 6.1.1. En ellos, en la posición `0x08`, mediante los caracteres `ASCII` se indica el tipo de compresión que se usa sobre los datos, `zlib`.

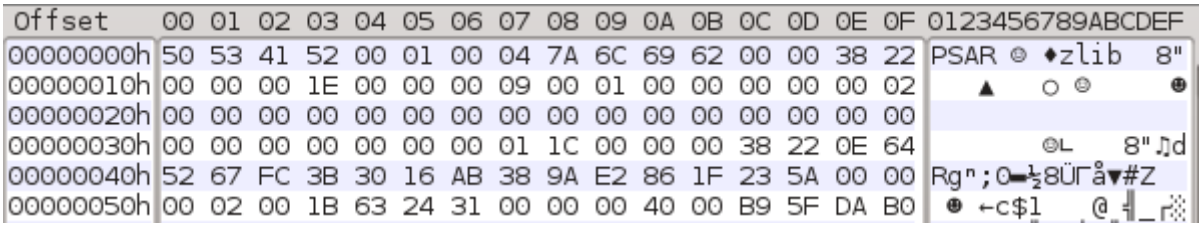
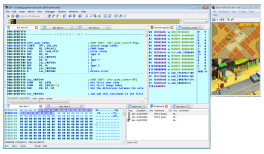
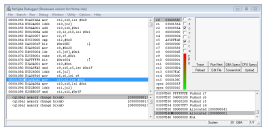


Figura 4.3: Primeros bytes del fichero `PSAR`.



(a) IDA Pro 6.1 con DeSmuME.



(b) Emulador con depurador integrado No$GBA.

Figura 4.4: Programas para depurar juegos de Nintendo DS.

4.2. Depuración de código

En ocasiones, el análisis de un fichero hexadecimal no es suficiente para estudiar un formato. Este es el caso de codificaciones complejas y estudios de cifrado e integridad en los que hace falta mirar la implementación del juego. En estos casos, mediante depuración del juego se pueden analizar las instrucciones máquina que ejecuta la consola, para saber cómo procesa el juego los ficheros y así poder estudiarlos. La depuración de juegos de Nintendo DS se puede realizar utilizando dos procedimientos (Figura 4.4).

El primer método es usar el emulador DeSmuME y el depurador IDA Pro. Este emulador se puede compilar activando funciones de depuración remotas. Incluye una implementación de *GDB Remote Serial Protocol*[2] que permite que programas externos controlen la ejecución del juego paso a paso y accedan a la memoria `RAM`. Esto es lo que hace IDA Pro, sirviendo como un depurador universal. De esta forma se puede visualizar el código en ensamblador, poner puntos de interrupción y ver y cambiar la memoria del juego.

La alternativa es usar el emulador privativo No$GBA. Existe una versión que incluye un depurador originalmente de pago, pero que recientemente se liberó como *freeware*. Incluye las mismas funciones descritas anteriormente, pero en este caso implementadas sobre el propio emulador, aumentando así la eficiencia.

Dado que IDA Pro es un programa con una licencia básica de coste $500, este trabajo se realizó con el emulador No$GBA que, a pesar de no tener tanta funcionalidad, sí provee de las necesarias para realizar el trabajo sin coste.

4.2.1. Búsqueda de archivos en memoria RAM

Esta subsección explicará el procedimiento seguido para, mediante depuración, averiguar cómo el juego procesa un fichero. Para ello hace falta conocer cómo se obtienen datos del cartucho del juego. El protocolo de comunicación es sencillo: cuando se necesitan datos, se envía un comando que el hardware cifra (esa parte, al usar un emulador, no se realiza), y al cual al cartucho responde con los datos solicitados. El código de este comando es `0xB7` y se envía escribiéndolo en el puerto virtual `0x040001A8` [15].

Esta dirección se puede buscar en el código ensamblador, encontrando la función que solicita los datos. Gracias a esto se puede poner un punto de interrupción y comprobar qué direcciones se piden. Dado que continuamente se están cargando datos, realizar esto manualmente no es viable. Es por ello que, aprovechando que el emulador DeSmuME es de código abierto, se puede modificar para automatizarlo.

Esta mecánica se implementó para realizar los análisis de este trabajo, desarrollando el programa *NitroFilcher*[3]. En el emulador se añadieron las siguientes líneas de código a la función `HandleDebugEvent_Execute` del archivo `debug.cpp`, consiguiendo que cada vez que se ejecutara las instrucciones que solicitan datos, guardase en un fichero de texto la posición que se pedía junto al tamaño:

```
// 0x02016DE0 es la dirección de la función que solicita datos.
if (DebugEventData.addr == 0x02016DE0 && log_ptr != NULL) {
```

[2] `http://www.embecosm.com/appnotes/ean4/embecosm-howto-rsp-server-ean4-issue-2.html`
[3] `https://github.com/pleonex/AiroRom/tree/master/Programs/NitroFilcher`

```
    u32 addr = DebugEventData.cpu()->R[2];
    u32 size = DebugEventData.cpu()->R[3];
    fprintf(log_ptr, "%08X,%08X,%08X\n", addr, size, addr + size);
}
```

Este fichero luego lo procesaría el programa *NitroFilcher*, escribiendo en otro fichero de texto las rutas a los archivos que correspondían esas direcciones.

4.2.2. Búsqueda de algoritmos sobre textos

En el Capítulo 5 se mostrarán algoritmos para ofuscar y cifrar textos. El procedimiento para realizar este estudio se basa en depuración de código.

El primer paso es buscar la frase en texto plano sobre el binario del juego. Tras determinar que no aparece, se puede confirmar que puede estar codificada en una codificación no estándar, comprimida o cifrada. Se suele probar con las codificaciones soportadas nativamente por las fuentes de Nintendo DS, como son `UTF-8`, `UTF-16`, `SHIFT-JIS` y `CP1252`.

Confirmado que hay un algoritmo que se aplica sobre el texto, el objetivo es encontrar este texto en la memoria `RAM` del juego, para poder poner un punto de interrupción y ver qué transformaciones se aplican hasta llegar al texto descifrado. Para ello, se extrae la memoria justo en el momento de mostrar un diálogo de texto, pues debería estar la frase guardada para poder mostrarla en pantalla. Esto se puede realizar con cualquier emulador de Nintendo DS. Una vez extraído el archivo binario con la memoria, se busca el texto mediante visores hexadecimales, usando de nuevo codificaciones estándar. Si la frase se encuentra, se pondría un punto de interrupción en dicha posición y se estudiaría el algoritmo. De no ser así, probablemente se use una codificación propietaria. En esos casos se puede encontrar siguiendo la metodología explicada en los siguientes apartados.

Búsqueda de la tipografía

El primer procedimiento consiste en buscar el archivo con la tipografía del juego, pues la codificación será la misma que el orden con el que aparecen los caracteres en ella. El estándar de este formato tiene la cabecera `NTFR` y existen programas para extraer y editar este tipo de formatos[4].

Búsqueda diferencial

El segundo procedimiento pasa por desarrollar un programa de búsqueda diferencial. Una implementación se llevó a cabo para este estudio, denominado *RelativeSearch* y disponible en el repositorio del trabajo[5].

La idea de este programa se basa en que un grupo de caracteres similares tiene una numeración seguida. Por ejemplo, si la codificación indica que el carácter 'a' tiene asociado el valor `0x0123`, el carácter 'b' tendrá el siguiente, `0x0124`. Restando ambos valores, se encuentra su 'distancia' en el abecedario. Esta distancia, en la mayoría de los casos, será la misma en cualquier codificación, incluso propietaria, a no ser que se hayan desordenado los caracteres.

RelativeSearch (Figura 4.5) implementa esta idea. Para ello dada una palabra como 'hola' resta cada carácter sobre el anterior y guarda esa diferencia. A continuación, realiza esta misma operación sobre un fichero binario, resta un byte sobre el anterior y compara la diferencia. Si la diferencia coincide, se muestra en pantalla la posición de la coincidencia. El programa soporta la búsqueda de codificaciones de 8 y 16 bits.

[4] `https://github.com/pleonex/NerdFontTerminatoR`
[5] `https://github.com/pleonex/RelativeSearch`

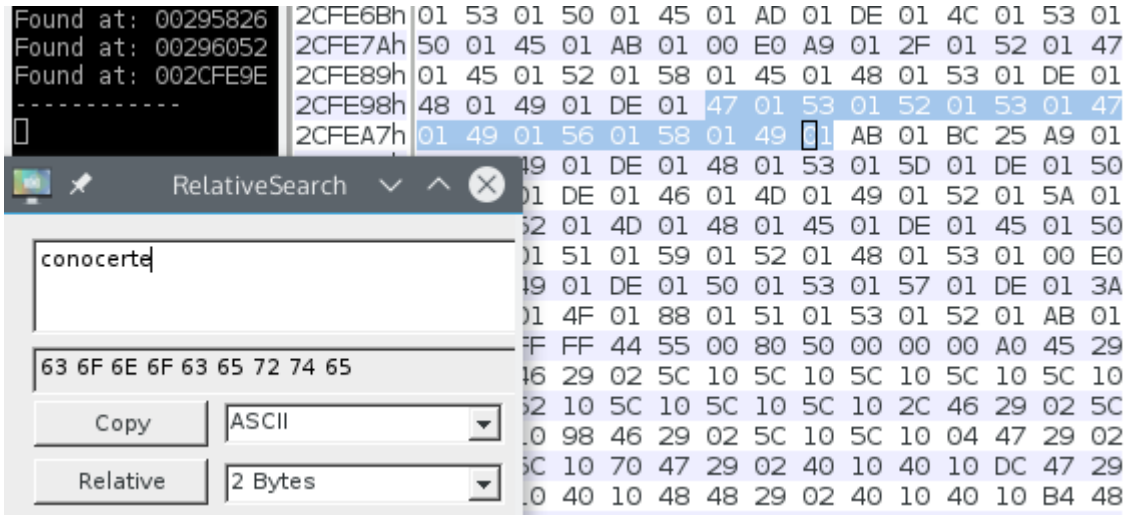


Figura 4.5: Frase con codificación no estándar encontrada por RelativeSearch en *Pokémon*.

4.2.3. Búsqueda de algoritmo sobre imágenes

El procedimiento para encontrar el algoritmo de cifrado en este caso es distinto al realizado con texto. No se puede partir de datos conocidos como anteriormente se hacía con una frase que se veía en la pantalla. Sin embargo, dado que las cabeceras de la imagen suelen ser estándar, se puede buscar sobre el código en ensamblador la función que la procesa. En concreto se busca el identificador `CHAR`, que se lee para determinar el comienzo de una sección del archivo.

Una vez encontrada la función que carga una imagen, se puede poner un punto de interrupción sobre ella para tener acceso a los datos cifrados cuando el juego procese un archivo. Una vez localizados, se añade un punto de interrupción sobre los datos cifrados de la imagen, el cual llevará al código que los procesa y, por tanto, el algoritmo que los descifra.

4.3. Interceptación de la comunicación

La Nintendo DS fue la primera portátil de Nintendo en soportar servicios en línea mediante una conexión Wi-Fi. Esta comunicación se ha estudiado en el Capítulo 7, explicándose en esta sección la metodología seguida para capturar los paquetes.

Para ello se preparó un escenario para realizar un ataque *man-in-the-middle*. Un punto de acceso se conectó al ordenador mediante *Ethernet* y este, a su vez, mediante conexión Wi-Fi, con el punto de acceso original. Dado que el sistema operativo *Fedora 20* tiene por defecto una configuración restrictiva de red, se tuvo que diseñar el siguiente *script* para configurar una subred mediante *NAT* y permitir el redireccionamiento de paquetes:

```
# El primer argumento es la IP de la interfaz que tiene conexión a Internet
# Crea la ruta hacia la subred
sudo route add -net 192.168.3.0/24 gw 10.42.0.29 p35p1

# Habilita SNAT para la segunda subred
sudo iptables -t nat -I POSTROUTING -s 192.168.3.0/24 -o wlp4s0 -j SNAT --to $1

# Permite los paquetes de destino la segunda subred
sudo iptables -t filter -I FORWARD -d 192.168.3.0/24 -j ACCEPT

# Permite los paquetes con origen la segunda subred
sudo iptables -t filter -I FORWARD -s 192.168.3.0/24 -j ACCEPT
```

```
# Permite sin restricción los paquetes con destino la primera subred
sudo iptables -t filter -I FORWARD -d 10.42.0.0/24 -j ACCEPT
```

Usando *Wireshark* para capturar los paquetes sobre la interfaz de *Ethernet* se pudo ver todo el tráfico. Este escenario requería hardware y era complejo de montar cada vez que se quería analizar un juego. Es por ello que, aprovechando que el emulador DeSmuME soporta las comunicaciones Wi-Fi y es de código libre, se modificó para guardar los datos que transmitía y enviaba en formato *PCAP*, compatible con *Wireshark*.

Se modificaron los archivos `wifi.cpp` y `wifi.h`[6], añadiendo los siguientes métodos:

- `create_packet()`: inicializa un nuevo archivo `PCAP` para guardar los datos. Esta función se llama cada vez que se realiza una petición de asociación con el punto de acceso, es decir, cuando se inicia una conexión.
- `save_packet(u8* packet, u32 len)`: guarda un paquete *Ethernet* en el archivo.
- `save_adhocPacket(u8* packet, u32 len, void* addrGen, bool isSent)`: genera un paquete con cabeceras *Ethernet*, *IP* y *UDP* y guarda los datos de la capa aplicación que se le pasa. Esta función se llama cuando se envía un paquete mediante comunicación *ad-hoc* a otra consola. Dado que este paquete solo contiene datos en formato del protocolo de Nintendo, para poder analizarlo con *Wireshark* es necesario crear las cabeceras de la capa de enlace, red y transporte.

Esta comunicación sin embargo está cifrada. En la Sección 7.1.1 se explica, sobre los resultados de los servicios en línea de Nintendo, cómo utilizar los programas desarrollados *RC4Finder* y *SSLPatcher* para capturar una comunicación descifrada.

4.4. Documentación y repositorio

Los programas y documentación de este trabajo se han ido publicando en un repositorio, gestionado por *git*, y subido a la página *GitHub*, con el siguiente enlace:

`https://github.com/pleonex/AiroRom`

La documentación se fue redactando en formato *Markdown* en la wiki[7] del repositorio. En la página de *Mecanismos a investigar*, se muestran los juegos ordenados por desarrollador con más juegos publicados. Para crear esta clasificación se hizo el *script* en *python* llamado *Scenadvorter*. Este programa lee un XML con una base de datos de todos los juegos para Nintendo DS que proporciona la página ADVANsCEne[8], y clasifica los juegos por desarrollador mostrando los diez primeros con más juegos.

Para guardar la información sobre los algoritmos descubiertos se desarrolló un programa que permitía guardar y editar dicha información en ficheros XML. Este programa (Figura 4.6) se llamó *DataBrithm* y su código se encuentra en el repositorio también.

En la carpeta `Games` y `Programs` del repositorio se hallan los programas desarrollados para analizar los contenidos de los juegos. Esta memoria, realizada con LaTeX [16], está en `Report`.

[6] `https://github.com/pleonex/AiroRom/tree/master/Programs/DeSmuME%20PCAP`
[7] `https://github.com/pleonex/AiroRom/wiki`
[8] `http://www.advanscene.com`

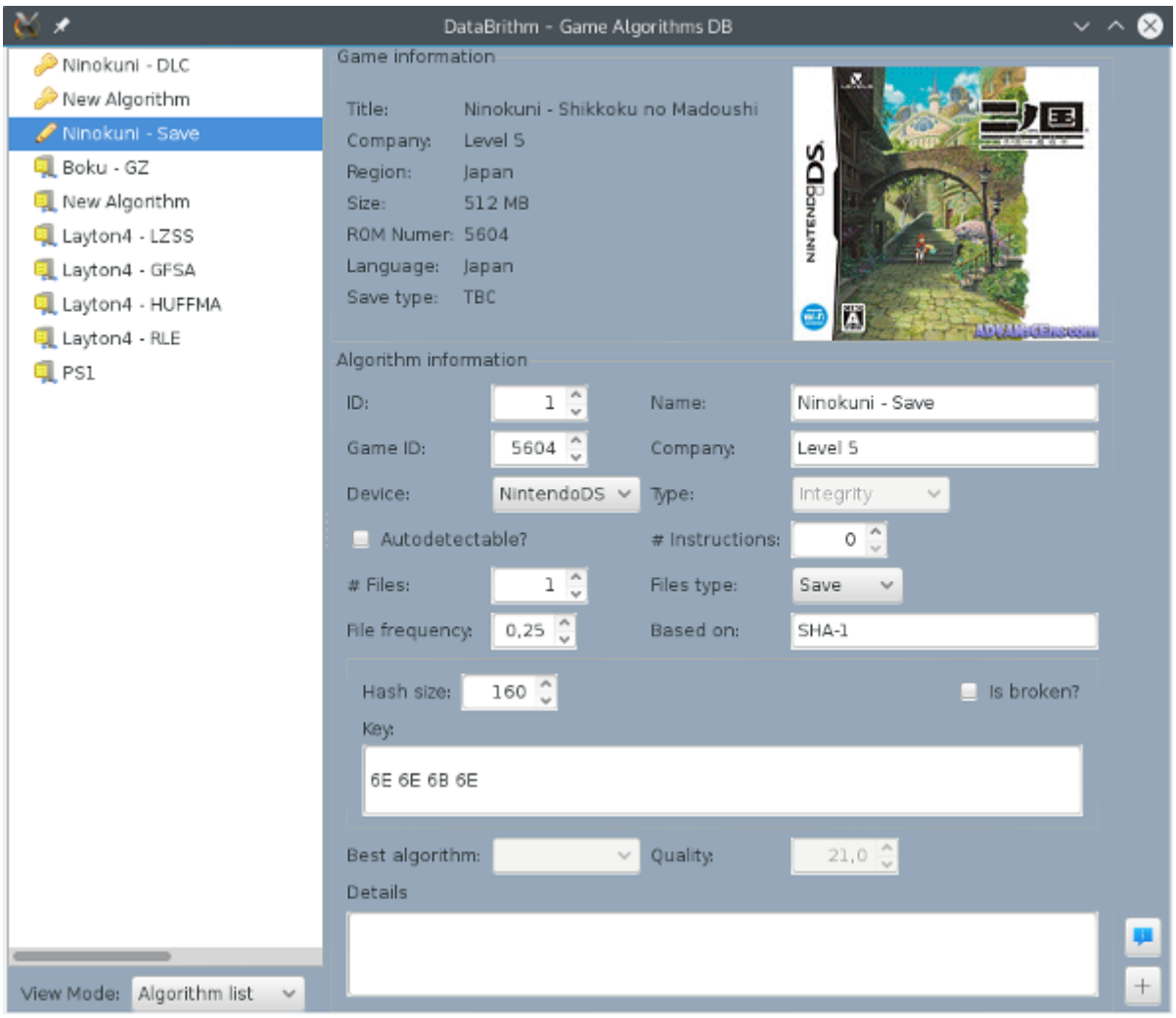


Figura 4.6: Programa de gestión de algoritmos de juegos DataBrithm.

Capítulo 5

Traducciones no oficiales

Las traducciones no oficiales, conocidas informalmente como *fan-traducciones* [27], son trabajos realizados por personas externas a la compañía desarrolladora. Su objetivo es traducir un videojuego y compartirlo de forma altruista. La distribución se hace comúnmente en foros dedicados y mediante *parches*, es decir, archivos binarios que contienen solo las modificaciones realizadas y, en principio, no contenido con derechos de autor. Este capítulo se ha centrado en los resultados obtenidos al analizar juegos principalmente de la consola Nintendo DS. La selección se ha realizado en base a sagas que han tenido más traducciones no oficiales (*Pokémon*), juegos con proyectos largos de traducción (*Ninokuni*) y aquellos en los que se han visto solicitudes o intentos (*London Life* y demás).

Este tipo de trabajos se remontan al origen de los ordenadores cuando, en 1993, se forma el primer grupo de traducción llamado Oasis para MSX [24]. A partir de entonces, han surgido numerosos proyectos, con dos páginas webs como referencia: GbaTemp.net[1] y ROMHacking.net[2]. El motivo para iniciar un proyecto es la confirmación por parte de una empresa de que no se llevaría a cabo la traducción del juego. Como se explica en el último apartado, a pesar de ello han existido casos en los que la traducción se ha iniciado antes de dicha confirmación y tras producirse esta, esos proyectos se detuvieron. No ha sucedido así con todos los juegos, pues en la saga de *Pokémon* han existido traducciones parciales antes de la oficial, perjudicando las ventas de la compañía, pues los usuarios prefieren obtener una copia no lícita del juego antes de esperar a la salida oficial. Es en estos juegos donde, como se ha analizado en la siguiente sección, se observa una evolución de mecanismos para evitar y retrasar estos trabajos.

Estos hechos se han visto reflejados recientemente en dos cartas de *cese y desista* enviadas por la empresa *Square Enix* a los grupos de traducción de *Final Fantasy Type-0* [21] para *Play Station Portable* (PSP) y *Dragon Quest VII* [2] para Nintendo 3DS. En ellas se alega que por 'motivos de mercado' no pueden permitir esos trabajos, refiriéndose a la salida de estos títulos para otras consolas. En ambos casos, los grupos de traducción pararon la traducción.

Relacionado con las traducciones, se encuentran los *mods* [29], modificaciones sobre un videojuego con el objetivo de crear una versión alternativa. Usando el motor del juego se cambia historia, imágenes, sonidos y *scripts*. Existen comunidades dedicadas exclusivamente a este motivo como es *Whack a Hack!*[3] para la saga *Pokémon*.

Los mecanismos que se van a comentar a continuación muestran cómo los archivos con contenido de texto e imágenes son los más protegidos u ofuscados. Esto no evita en su totalidad la edición, pues, teniendo acceso al código máquina que ejecuta la consola, se puede mirar cómo accede a los ficheros. Sin embargo, sí puede retrasar o al menos imposibilitar para aquellas personas que no tengan los conocimientos técnicos necesarios.

[1] `http://gbatemp.net/`
[2] `http://www.romhacking.net/`
[3] `http://wahackpokemon.com/es`

5.1. Saga Pokémon en Nintendo DS

Pokémon es una serie de juegos desarrollados por *Game Freak* desde 1993 para las consolas de Nintendo. La saga se divide en denominadas 'generaciones', que engloban al menos tres juegos. Ha sido una de las franquicias más exitosas a nivel mundial [5], con series de animación y películas entre otros.

A causa de la fama de los juegos y la falta de paciencia de los seguidores, se desarrollan traducciones parciales del juego. Estos proyectos, comenzados pocas horas después del primer lanzamiento en Japón, pretenden ofrecer una versión parcialmente traducida en semanas, adelantándose a los lanzamientos oficiales. Este es el caso de comunidades como PokéCheats.net[4] o ProjectPokemon.org[5].

Es por estos motivos que *Game Freak*, desde el primer juego de Nintendo DS, incluyó métodos de ofuscación, con el objeto de no solo evitar traducciones, si no también los *mods*. Los textos que aparecen en diálogos y menús, al igual que las imágenes principales de *Pokémons*, son el primer objetivo a la hora de realizar una traducción o edición. Estos han sido el objeto de estudio y se han encontrado los algoritmos de protección que se analizarán en los siguientes subapartados. Además, se ha podido comprobar una evolución de estas técnicas a lo largo de las generaciones.

Los sonidos y música han sido los únicos archivos que no han estado protegidos. Se encuentran en un formato y codificación estándar con extensión `sdat` en la carpeta raíz del juego.

5.1.1. Pokémon Perla y Diamante

Los juegos *Pokémon Diamante y Perla* corresponden a la cuarta generación de saga y a la primera que salió para Nintendo DS. Primero llegó a Japón en septiembre de 2006, siete meses después a los Estados Unidos y tres meses después a Europa.

Archivos

Una de las características de la Nintendo DS respecto a su antecesora (GameBoy Advance) es la introducción del sistema de ficheros. El formato del juego incluye una sección en la que indicar cómo dividir los datos en ficheros y clasificarlos en carpetas con nombres. Esto ha facilitado la tarea de investigación ya que, en la mayoría de los juegos, los archivos tienen nombres descriptivos sobre su contenido.

Es el caso de esta generación donde, como se ha visto (Figura 5.1), la jerarquía de carpetas y aún más el nombre de los archivos ofrecen pistas de su contenido. Por ejemplo, en `msgdata/msg.narc` se han encontrado los textos del juego, en `itemtool/itemdata/item_icon.narc` las imágenes de los objetos y en `graphic/font.narc` las tipografías del juego.

Textos

La investigación de los textos se llevó a cabo usando la metodología descrita en la Sección 4.2.2. Se ha podido comprobar que están cifrados, además de codificados sin usar un estándar (como podría ser `ASCII` o `UNICODE`). Para usar el primer método descrito, se ha buscado y estudiado la tipografía del juego, determinando así la tabla de codificación, es decir, el valor número que está asociado a cada carácter (Tabla 5.1).

El algoritmo es un cifrado con operación `XOR` y clave dinámica aplicado sobre cada byte individualmente. La clave inicial es igual a `clave = (ushort)(0x91BD3 * (i + 1))`, donde `i` es el bloque de texto a descifrar. Después de descifrar un byte se actualiza según `clave = (ushort)(clave + 0x493D)`.

[4] `http://pokecheats.net/tools/translations.php`
[5] `http://projectpokemon.org/`

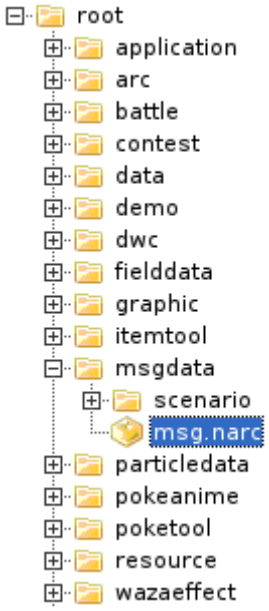


Figura 5.1: Sistema de ficheros en Pokémon Perla visto con Tinke.

Tabla 5.1: Especificación de tipografías en Pokémon Perla.

Posición	Tamaño	Descripción
		Cabecera
`0x00`	`0x04`	Puntero a la sección de datos
`0x04`	`0x04`	Puntero a la sección VWT
`0x08`	`0x04`	Número de caracteres
`0x0C`	`0x01`	Ancho de un carácter
`0x0D`	`0x01`	Alto de un carácter
`0x0E`	`0x01`	BPP
`0x0F`	`0x01`	BPP
		Datos de la fuente[a]
		Variable Width Table (VWT)[b]

[a] Los caracteres están codificados por *tiles* de 8×8 píxeles en formato horizontal.
[b] Cada byte indica el ancho del carácter.

El cifrado se ha aplicado también sobre los campos de la tabla de contenidos (ver Tabla 5.2). Se usa la misma clave de 16 bits, concatenada dos veces, para aplicarla sobre los valores de 32 bits *posición* y *tamaño* de una misma entrada. Se inicializa con: `clave = (ushort)(clave_base * 0x2FD * (i + 1))`, donde `i` es el número de la entrada a descifrar. Como se ve, se hace uso de una *clave base* que se encuentra disponible en la cabecera del fichero.

Sin embargo, esta parte del fichero presenta un fallo de seguridad, pues se está guardando la clave en texto plano. Como se ha comentado los valores cifrados son de 32 bits (4 bytes) pero, en el caso del campo *posición*, la mayoría de las veces **solo se usan los dos primeros bytes**. Ello se debe a que, como el tamaño del fichero de texto no supera los `2^16 = 65535 bytes`, no hacen falta posiciones más grande de estas. Esto ocurre también sobre el campo *tamaño* donde se aplica la misma clave, ya que **en pocas ocasiones un diálogo ocupará más de 65 KB.** Debido a que la clave se concatena para descifrar el valor de 32 bits, se está aplicando sobre bytes con valor 0, quedando la clave almacenada en texto plano: `valor_cifrado = (00 00 AB CD) XOR (XW YZ XW YZ) = (XW YZ EF GH)`, siendo `00 00 AB CD` el campo descifrado y `XW YZ` la clave.

En la Tabla 5.2 se especifica el formato de los archivos de texto.

Tabla 5.2: Especificación del formato de texto en Pokémon Perla.

Posición	Tamaño	Descripción
		Cabecera
`0x00`	`0x02`	Número de bloques de texto
`0x02`	`0x02`	Clave base para descifrar TOC
		Table Of Content[a]
`0x00`	`0x04`	Posición del texto[b]
`0x04`	`0x08`	Tamaño del texto[b]
		Texto[b]

[a] Se especifica solo la primera entrada.
[b] Campo cifrado descrito en la Sección 5.1.1.

Imágenes

En este juego las imágenes no contienen texto, por lo que no son necesarias editarlas en una traducción. Sin embargo, son uno de los recursos a editar para realizar un *mod*. Es por ello que se ha podido ver cómo las imágenes del archivo `poketool/pokegra/pokegra.narc`, que contiene seis *sprites* por *Pokémon* usados durante las batalla, están cifradas. El cifrado se realizó sobre los datos de la imagen, manteniendo las cabeceras del formato.

Aplicando la metodología descrita en la Sección 4.2.3 se pudo encontrar el algoritmo. Usando bloques de datos con tamaño fijo de `0x1900` bytes, aplicar la operación `XOR` sobre valores de 16 bits comenzando por el final. La clave es de 32 bits y se actualiza después de usarse. Se inicializa con el primer valor cifrado, últimos dos bytes del fichero, e irá cambiando según `clave = (uint)(clave * 0x41C64E6D + 0x6073)`.

Conclusión

Puntos fuertes

- Los textos usando una codificación no estándar.
- Los textos están cifrados con una clave dinámica.
- Las imágenes están cifradas con un clave dinámica.

Puntos débiles

- Los archivos y carpetas tienen nombres descriptivos.
- Hay un fallo de seguridad en el cifrado de la tabla de contenido de los textos por el que se puede averiguar la clave.
- Las imágenes usan codificaciones estándar por lo que el software existen de procesamiento se puede utilizar una vez descifrado el fichero.

5.1.2. Pokémon HeartGold y SoulSilver

Los juegos *Pokémon HeartGold y SoulSilver* son una remasterización para Nintendo DS de la segunda generación, originalmente para GameBoy Advance. Se lanzó primero en Japón en septiembre de 2009 y en Estados Unidos y Europa seis meses después. No hubo mucha diferencia entre los lanzamientos de Estados Unidos y Europa, por lo que no se realizaron traducciones no oficiales desde el inglés, solo desde el japonés que llegó a completarse al 90 %[6].

[6] En el siguiente hilo se llevó a cabo el proyecto: `http://projectpokemon.org/forums/showthread.php?4534-Pokemon-HeartGold-and-SoulSilver-Translation-Patch`

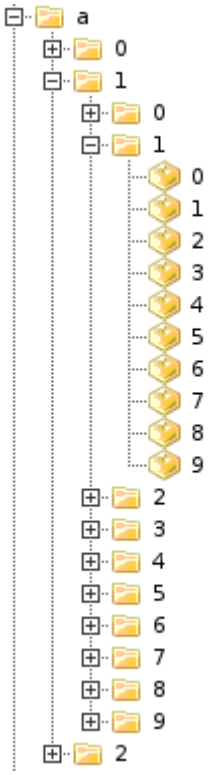


Figura 5.2: Carpetas y ficheros ofuscados en Pokémon HeartGold.

Archivos

La primera gran diferencia en estos nuevos juegos llegó con el ofuscado de nombres de archivos y directorios. Como se observa en la Figura 5.2, todos los archivos principales del juego están nombrados con números, al igual que las carpetas que los contienen. Además, el contenido de estas carpetas no relaciona a los archivos entre sí generando una organización aleatoria. En total hay 275 archivos que, a su vez, son contenedores de 1000 ficheros sin nombre pero relacionados en contenido, como son los textos o *sprites* de los *Pokémons*.

Analizando el código en ensamblador se pudo encontrar la forma interna en la que se accede a estos ficheros. Se realiza llamando a varias funciones de similares, el primer parámetro es un ID que identifica al archivo y el segundo el índice del fichero dentro del contenedor. Este ID es lineal y si se descompone en base diez se puede calcular la ruta absoluta. De esta forma, el ID `0x1B (27)` corresponde a `27 = 0*100 + 2*10 + 7*1`, lo que significa que estaría ubicado en `a/0/2/7`. A la hora de implementar este mecanismo durante el desarrollo del videojuego se podría haber usado un archivo de cabecera similar al siguiente:

```
file_paths.h
#define MESSAGE_FILE_ID 0x1B
#define POKEMON_IMAGES_FILE_ID 0x1C
```

donde cada constante corresponde a un fichero, facilitando el desarrollo pero ofuscando el producto final. Para poder editarlos, habría que investigar archivos, sin nombres, binarios y sin estructuras de datos conocidas, mediante depuración de código, sin contar con pistas como pasaba en la edición anterior donde el nombre era descriptivo del contenido.

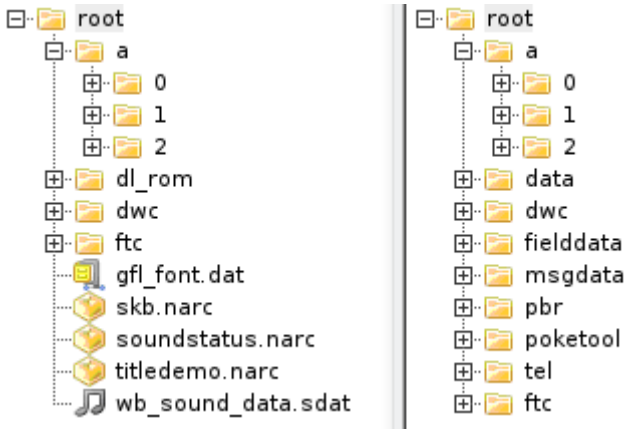


Figura 5.3: Comparación de carpetas entre Pokémon Blanco (izq.) y Pokémon HeartGold (der.).

Textos

Tras realizar una búsqueda del algoritmo, que se aplica en los textos (Sección 4.2.2), se pudo comprobar que están cifrados. Además, tanto el formato del archivo, el algoritmo de cifrado y las claves son las mismas a las descritas en los juegos anteriores (apartado textos de la Sección 5.1.1). El formato de las fuentes es también el mismo.

A pesar de que el cifrado no cambió y que los mismos programas de edición se podían usar, se ofuscó este archivo pasando a estar en `a/0/2/7`.

Imágenes

Al igual que sucedió con los textos, tras encontrar el algoritmo de cifrado de imágenes (Sección 4.2.3), se descubrió que se utiliza el mismo algoritmo de cifrado con las mismas claves. Solo existe una diferencia de implementación, ya que el descifrado empieza por el comienzo del fichero en lugar de por el final.

Conclusión

Puntos fuertes

- Los nombres de los archivos y directorios están ofuscados.

Puntos débiles

- No todos los archivos están ofuscados como se ve en la Figura 5.3.
- Se mantiene los mismos algoritmos y claves que en la generación anterior por lo que, una vez encontrado los archivos, se pueden reutilizar las mismas herramientas de edición.

5.1.3. Pokémon Blanco y Negro

La quinta generación corresponde a los juegos *Pokémon Blanco y Negro* para Nintendo DS. Salieron en Japón en septiembre de 2010 y en marzo de 2011 en Estados Unidos y Europa. Hubo un proyecto de traducción desde el japonés al inglés[7] que llegó a estar casi completado.

Archivos

Los archivos y carpetas fueron ofuscados al igual que pasó con la generación anterior. En este caso, se protegieron todos los archivos en lugar de los principales como se ve en la Figura 5.3. Los únicos archivos que se dejaron fuera fueron el de sonido y la demo.

[7]En el siguiente foro se puede encontrar el desarrollo del proyecto: `http://projectpokemon.org/forums/showthread.php?11397-Pokemon-Black-and-White-Translation-Project`

Tabla 5.3: Especificación del formato de texto en Pokémon Blanco.

Posición	Tamaño	Descripción
		Cabecera
`0x00`	`0x02`	Constante: `0x01`
`0x02`	`0x02`	Número de bloques de texto
`0x04`	`0x04`	Tamaño de la sección de datos
`0x08`	`0x04`	Constante: `0x00`
`0x0C`	`0x04`	Tamaño de la cabecera
		Datos
`0x00`	`0x04`	Tamaño de la sección
`0x04`	`0x04`	Posición relativa a la sección[a]
`0x06`	`0x02`	Número de caracteres[a b]
`0x08`	`0x02`	Desconocido[a]
		Texto[c]

[a] Se especifica solo la primera entrada.
[b] Cada carácter es un valor de 16 bits.
[c] Campo cifrado descrito en la Sección 5.1.3.

A pesar de usar internamente la misma forma de acceso que se describió en el apartado de archivos de la Sección 5.1.2, los identificadores, y por tanto localización de estos, cambió. Por ejemplo, el archivo con los textos pasa a estar en `a/0/0/2` en lugar de `a/0/2/7`.

Textos

En los dos apartados anteriores (5.1.1 y 5.1.2), se describió cómo los textos usaban el mismo algoritmo y clave para cifrarse. Una vez encontrado el algoritmo de esta generación (Sección 4.2.2), se pudo ver que se trata de un cifrado `XOR` con clave dinámica pero cambia respecto a las pasadas ediciones.

Una de las principales diferencias es que no se usa una codificación propia, que dificulta la investigación, si no que se usa el estándar UTF-16. La clave inicial se genera según `clave = (i + 3) * 0x2983`, donde `i` es el número de bloque al que se quiere acceder. A continuación se aplica la operación `XOR` sobre un carácter (un valor de 16 bits ya que se usa UTF-16) y se actualiza la clave. Para ello, se mueven los tres bits más significativos a la posición menos significativa:

```
temp  = clave & 0x1FFF;              // Elimina los bits más significativos
clave = (temp << 3) | (clave >> 13) // y los añade al principio.
```

Al desplazarse un número impar de veces, cada bit pasará sobre cada posición de la clave una vez, obteniéndose el máximo de permutaciones. En una clave de 16 bits, esta será de 16.

La estructura de datos se especifica en la Tabla 5.3, donde se puede ver que la sección con la tabla de contenido no está cifrada a diferencia de cómo pasaba en ediciones anteriores.

Imágenes

Las imágenes, a diferencia de las pasadas ediciones, no se han cifrado. En cambio, no se utiliza un formato estándar de codificación excepto para las paletas. Existe un conjunto de ficheros adicionales con nuevos formatos con la información necesaria para recomponer la imagen (Figura 5.4).

El cambio de formato es más eficaz como método de protección para prevenir modificaciones. Aunque podría ser que el único motivo fuera técnico (representar animaciones), usando un nuevo formato hace que los programas de procesado de imagen existentes no se puedan usar. Esta tarea requiere de investigación de formatos y de programación, se requiere más esfuerzo y tiempo que crear un decodificador.

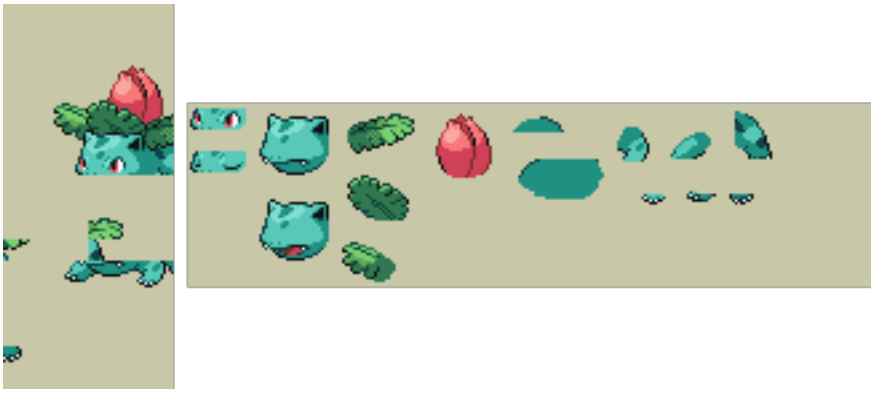


Figura 5.4: Imágenes sin cifrar en Pokémon Blanco.

Conclusión

Puntos fuertes

- Los nombres de las carpetas y archivos están ofuscados.
- Los textos están cifrados con un nuevo algoritmo y clave.
- Las imágenes usan archivos extras con especificaciones nuevas para recomponerlas por lo que software de procesamiento nuevo sería necesario.

Puntos débiles

- Los textos no usan una codificación propia.
- Las imágenes no está cifradas.

5.1.4. Pokémon Conquest

Pokémon Conquest es un juego desarrollado por *Tecmo Koei* y publicado en el año 2012 para la consola Nintendo DS. Este juego dista de la mecánica de la saga principal al ser de combates tácticos por turnos.

Aunque hubo una diferencia de tres meses entre la salida en Japón y Estados Unidos, la compañía no confirmó de inmediato las fechas. A consecuencia, se formó un grupo de traducción no oficial[8] que llegó a casi completarse.

Archivos

En este juego no se han ofuscado los archivos y carpetas, teniendo nombres descriptivos de su contenido.

Textos

Los textos de este juego están cifrados como se ha podido ver tras encontrar el algoritmo que los descifra y decodifica (Sección 4.2.2). En concreto, primero descifra un bloque de datos y luego lo descodifica para poder representar dichos caracteres con la fuente del juego. La codificación no se usa por la fuente (esta implementa el estándar *Latin-1*), es una capa adicional para ahorrar espacio y posiblemente ofuscar.

[8]Foro donde se desarrolló el proyecto:
`http://pokecheats.net/forum/showthread.php?13925-Translation-project-for-Pokémon-Nobunaga-s-Ambition`

Tabla 5.4: Especificación del formato de texto en Pokémon Conquest.

Posición	Tamaño	Descripción
Table Of Content[a b]		
0x00	0x04	Posición del bloque
0x04	0x04	Tamaño del bloque
Bloques[c]		

[a] Se especifica solo la primera entrada.
[b] Existen 33 bloques.
[c] Campo cifrado descrito en la Sección 5.1.4.

La codificación está optimizada para caracteres japoneses. Se basa en especificar solo un byte por carácter japonés en lugar de dos como usa la codificación SJIS. Para realizar esta reducción, se predice con códigos de control el primer byte del carácter, comprendido entre `0x81-0x83`, y que es igual entre caracteres cercanos. Se puede encontrar una implementación de esta codificación en el proyecto AmbitionText[9].

Respecto al cifrado, se basa en la operación `XOR` con la clave fija "`MsgLinker Ver1.00`" sobre cada byte de texto. A continuación se muestra la implementación en C# para el cifrado y descifrado[10]. El formato del fichero se especifica en la Tabla 5.4.

```
const string Key = "MsgLinker Ver1.00";
for (int i = 0; i < data.Length; i++)
    data[i] ^= (byte)Key[i % Key.Length];
```

Imágenes

Tanto los fondos como los *sprites* del juego no se han hallado cifrados. Sin embargo, se usan formatos de codificación específicos que suponen crear nuevo software para poder editarlos[11].

Conclusión

Puntos fuertes

- Los textos vienen cifrados y codificados.
- La codificación es única y difícil de implementar (más que el descifrado).
- Nuevos formatos de imágenes que requieren nuevos programas.

Puntos débiles

- Los archivos y carpetas no están ofuscados.
- Se usa una codificación final estándar que facilita la investigación.
- Las imágenes no están cifradas.

5.2. Ninokuni: El Mago de las Tinieblas

Ni no Kuni: Shikkoku no Madoshi (en español *Ninokuni: El Mago de las Tinieblas*) es el título de un juego que solo llegó a Japón desarrollado por *Level-5* para la consola Nintendo DS en 2010. La compañía confirmó que iba a ser traducido a otros idiomas debido a los problemas de traducción y distribución que suponía el libro físico que acompaña al juego [17].

[9] `https://github.com/pleonex/Pokemon-Conquest-Tools/AmbitionText`
[10] `https://github.com/pleonex/Pokemon-Conquest-Tools/Decrypted`
[11] `https://github.com/pleonex/tinke/tree/master/Plugins/PSL`

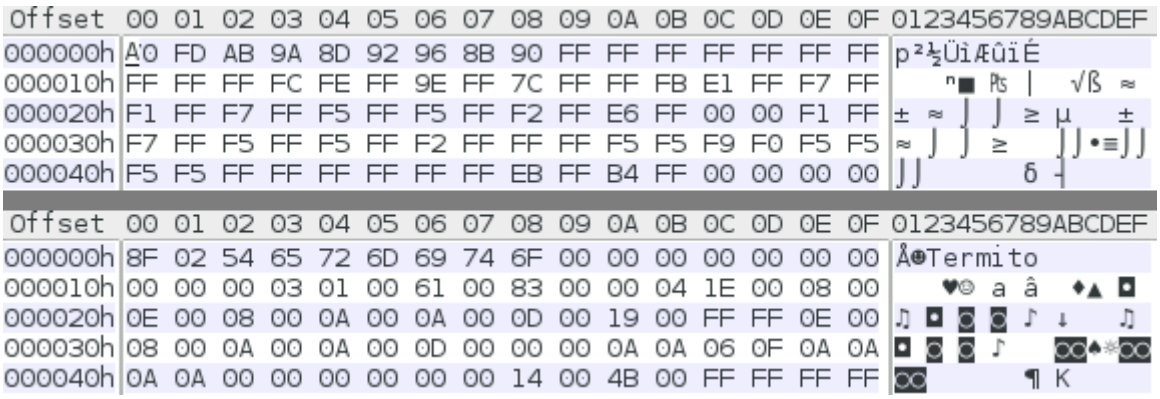


Figura 5.5: Archivo cifrado (arr.) y descifrado (ab.) de *Ninokuni: El Mago de las Tinieblas*.

A finales de 2011 se formó el grupo GradienWords[12] con el objetivo de realizar una traducción no oficial, directa del japonés al español. Este proyecto terminó el 29 de mayo de 2015, e incluye tanto un parche del juego en español como una copia escaneada del libro traducido en PDF.

Aparte de esta versión, existe una versión alternativa para Play Station 3 en varios idiomas incluyendo el español. A diferencia de la versión para Nintendo DS, solo el paquete *premium* incluye la copia física del libro y en inglés. Además, la historia y la mecánica del juego son diferentes.

La protección encontrada sugiere que su objetivo era evitar modificaciones del juego y no una traducción del mismo.

Archivos

Los archivos y carpetas no están ofuscados y tienen nombres descriptivos. Incluso se incluyen archivos de una versión anterior y de una *demo* que hubo del juego y que no se utilizan en la versión final.

Textos

En cuanto al apartado de textos hay que distinguir de dos tipos: *scripts* y descripciones.

Los *scripts*, debido a sus requisitos técnicos para ejecutar diferentes tipos de comandos con múltiples parámetros, son complejos de investigar y necesitan un análisis de las instrucciones máquina que los procesan. Solo existe un software que no ha sido publicado, desarrollado para la traducción al español del juego, que solo es capaz de editar los textos de los diálogos.

Respecto al segundo tipo, se trata de archivos con características de personajes, monstruos, objetos y ataques, con un campo de texto. El cifrado que se ha encontrado es básico, pero eficiente para evitar que en la mayoría de los casos pueda ser editado. Para descubrir el algoritmo no hizo falta analizar el código ensamblador, pues observando el fichero destacan las repeticiones del byte `0xFF` (Figura 5.5). Esto es un indicio de que se trata de la operación `XOR` con el byte (clave) `0xFF` o lo que es lo mismo, aplicar la operación `NOT` sobre cada byte.

Imágenes

Las imágenes no están cifradas pero usan una compresión privada sencilla. Sin embargo, esto requiere de nuevos programas para extraerlas e importarlas[13].

[12] `http://gradienwords.com`
[13] `https://github.com/pleonex/ninoimager`

Figura 5.6: Mensaje de error al detectar una partida modificada (izq.) y advertencia (der.) en Ninokuni.

Integridad en archivos de guardado

Adicionalmente se ha estudiado los mecanismos de protección sobre el archivo con los datos de guardado de la partida. Se trata de un fichero que en los cartuchos originales no se puede editar sin poseer hardware adicional, pero en una *flashcard* es fácilmente accesible al tratarse de un archivo de 64 KB.

Para evitarlo, en lugar de cifrar se ha implementado un algoritmo de integridad. En primer lugar se comprueba que el primer valor de 16 bits sea `0x0028` y que los últimos ocho bytes sean en hexadecimal `0B 3F A5 84 EC 32 9A 7D`.

A continuación, sobre el resto de los datos, desde la posición `0x0002` hasta `0xFFE4`, se calcula un *hash* usando el algoritmo `SHA-1`. Para añadir seguridad e imposibilitar la edición sin conocer todos los detalles de la implementación, en la posición `0xC5EC` se escriben solo durante el cálculo del *hash* los bytes `6E 6B 6E 6E` (en ASCII y *low-endian* equivalen a las consonantes del juego '*nnkn*'). Una vez realizado el cálculo y comprobación, estos bytes se ponen de nuevo a 0. Se trata de una clave que sin conocer ni su valor ni su posición, sería imposible calcular un código `SHA-1` válido. La suma de verificación se guarda en la posición `0xFFE4` del archivo y se comprueba durante el inicio del juego. En caso de no ser válido el mensaje muestra un mensaje de error y borra todos los datos de la partida.

Conclusión

Puntos fuertes

- Formato complejo en los *scripts*.
- Cifrado básico en algunos archivos de texto.
- Mecanismo de integridad en el archivo de guardado.

Puntos débiles

- Los *scripts* no están cifrados.
- Las imágenes no están cifradas.
- Dado que los archivos cifrados contienen muchos bytes de valor 0, la forma de descifrado queda visible.

5.3. Proyectos abandonados

Para terminar el capítulo se incluye un estudio sobre los proyectos de traducciones no oficiales que han sido abandonados, con sus respectivos motivos. El estudio se ha centrado en juegos para la Nintendo DS a partir de los datos de [12] y [23]. El objetivo será mostrar las causas más frecuentes por los que estos proyectos no se completan.

Ausencia de software de edición

Surgidos, por la falta de gente con conocimientos técnicos que puedan crear los programas necesarios de edición:

- WWII-Tank Battles (PS2): Ausencia de descompresor de archivos.
- Chi's Sweet Home: Chi ga Ouchi ni Yatte Kita! (NDS): Ausencia de importador de imágenes.
- Brave Fencer Musashi (PSX): Ausencia de editor de textos.
- Code Geass: Hangyaku no Lelouch (NDS): Ausencia de editor de textos.
- Custom Beat Battle Draglade 2 (NDS): Ausencia de editor de textos.
- Mobile Suit Gundam 00 (NDS): Ausencia de editor de textos.
- Super Robot Taisen OG Saga: Mugen No Frontier (NDS): Ausencia de editor de textos.
- Mysterious Dungeon: Shiren the Wanderer 2 (NDS): Ausencia de editor de textos y de modificaciones técnicas necesarias.

Problemas técnicos tras editar el juego

Problemas surgidos durante la traducción que no se llegaron a solucionar debido a falta de conocimientos o abandono de quienes crearon las herramientas:

- Dai Kaijuu Monogatari (PSX): Los programas usados para editar los textos causan bloqueos en el juego.
- Resident Evil 2 (DC y GC): El compresor no funcionaba con todos los archivos.

Abandono personal

Abandonos del grupo del proyecto por causas personales, falta de motivación y de tiempo:

- Professor Layton: London Life (NDS).
- Cross Treasures (NDS).
- Element Hunters (NDS).
- Hajime no Ippo The Fighting! DS (NDS).
- Naruto: Ninja Destiny 3 (NDS).
- Super Robot Wars K (NDS).

Traducción oficial

Abandonos de proyectos por la llegada o confirmación de una traducción oficial:

- Final Fantasy: The 4 Warriors of Light (NDS). Traducción inglesa.
- Fire Emblem: Shadow Dragon (NDS).
- Glory of Heracles (NDS).
- Inazuma Eleven (NDS).
- Kindom Hearts 356/2 Days (NDS).
- Naruto: Ninja Destiny 2 (NDS).
- Nostalgia (NDS).

El principal motivo de abandono es la ausencia de personas con conocimientos y dedicación para crear el software de edición necesario. La segunda causa más frecuente es la confirmación de la llegada de una localización, para no realizar un mismo trabajo dos veces y no dañar los intereses de mercado de la empresa.

Capítulo 6

Contenido con derechos de autor

La gestión de los derechos de autor ha generado un debate, que sigue sin estar resuelto, entre la libertad del usuario y la protección del creador de la obra [22]. Este incluye a los videojuegos, donde la piratería causa pérdidas millonarias [20]. De este contexto nace la idea del DRM, un mecanismo para imposibilitar la distribución no autorizada de contenidos.

Las características de estas técnicas se resumen en: detectar quién accede a cada obra, cuándo y bajo qué condiciones, autorizar o denegar de manera inapelable el acceso y autorizarlo bajo condiciones restrictivas [28]. En cuanto a mecanismos que se aplican para evitar la distribución ilegítima, los más comunes son [20]:

- Claves en CD. Se imprimía en la superficie del CD una clave que se pedía durante la instalación. La parte negativa es que si se perdía esta clave, el usuario legítimo no podía acceder, además del hecho de que con una simple búsqueda en Internet se podía obtener.
- Límite de instalaciones. Uno de los siguientes métodos fue controlar el número de instalaciones. Para ello, el producto se tenía que activar realizando una conexión con los servidores de la compañía. Sin embargo, esto requiere que el usuario tenga acceso a Internet y que los servidores estén activos. Además, limita el número de dispositivos del usuario.
- Conexión permanente (*Always-on DRM* [26]). Este método necesita una conexión en cada inicio del juego con los servidores de la compañía, e incluso en algunos casos durante toda la partida. Esto ha causado polémica debido a que el usuario no podría jugar sin conexión a Internet y porque los servidores tienen que estar activos todo el tiempo[1].
- Cifrado del software. Consiste en cifrar la aplicación usando claves asimétricas [14]. Como contrapartida, si el usuario pierde su clave privada y el acceso a ella, no podría volver a ejecutar ninguna aplicación.

El principal problema es la posibilidad de aplicar métodos de *ingeniería inversa* sobre el software autorizado. Una vez comprendido cómo una aplicación autorizada accede al contenido protegido, se podrían superar los mecanismos de protección para extraer, y en algunos casos modificar, el recurso. De aquí viene la necesidad de trabajar sobre entornos cerrados (Capítulo 2) cada vez más frecuentes en videoconsolas y móviles[2]. Una alternativa sería implementar el algoritmo en hardware como es el caso de la Nintendo 3DS.

[1] Algunos juegos de la compañía *Ubisoft* se quedaron sin acceso por tiempo indefinido debido a tareas de mantenimiento de los servidores: `http://www.gamespot.com/articles/ubisoft-drm-games-to-be-temporarily-unplayable/1100-6349732/`

[2] Los sistemas operativos iOS y Android no permiten el acceso al sistema de ficheros por defecto.

Aún encontrando un algoritmo eficiente en términos de protección y sin sobrecarga a la hora de procesado, una vulnerabilidad inevitable es el denominado ***agujero analógico***. La técnica consiste en reconvertir un medio digital en analógico, perdiendo los mecanismos DRM. Por ejemplo, grabar música que se está reproduciendo con un micrófono, grabar con una cámara una pantalla, escanear un libro o incluso copiarlo a mano. Estos procedimientos se utilizan en los videojuegos para, mediante funciones del emulador como capturas de pantalla o grabación de audio, extraer y compartir recursos en páginas como *Spriters Resource*[3].

En las siguientes secciones se analizarán los posibles mecanismos de seguridad que los juegos aplicarían sobre contenidos con derechos de autor. Para ello se han escogido una serie de juegos conocidos por tener materiales que frecuentemente son protegidos [30]. En primer lugar se analizarán juegos con libros electrónicos, a continuación se investigará el género *músical* y la protección sobre el material de vídeo. El resultado en todos los casos ha sido la carencia de cualquier mecanismo a la hora de proteger los contenidos, exceptuando en el material en formato de vídeo descrito en la última sección. También destaca la distribución de material adicional como el caso de un libro físico en *Ninokuni* para Nintendo DS o hardware en la saga *Guitar Hero: On Tour*.

6.1. Libros electrónicos

6.1.1. Ninokuni: La Ira de la Bruja Blanca

Ninokuni: La Ira de la Bruja Blanca es un juego desarrollado por *Level-5* para Play Station 3 en el año 2011. En el Capítulo de traducciones no oficiales (5.2) se expusieron los resultado del análisis de este juego para Nintendo DS. En esa versión se incluía un libro físico necesario durante el desarrollo del juego. Sin embargo, en Play Station 3 se incluye digitalmente y se puede acceder a él mediante un visor integrado en el juego.

El libro se encuentra en el archivo `hd05_es.sdat`, dentro de la carpeta raíz y en formato `PSAR`. Este tipo de formato, común en juegos de PS3, permite agrupar varios archivos y comprimirlos. La compresión, `zlib`, se indica claramente con cuatro caracteres en la posición `0x08` (Figura 4.3). Respecto a la estructura del fichero (Tabla 6.1), tiene una pequeña cabecera a la que le sigue una tabla de contenidos y los datos. En GitHub hay repositorios con programas para la extracción de los archivos[4].

Descomprimiendo el fichero se encuentran nueve archivos, de los cuales destaca uno de 438 MB en `data/book/es/BigData/gdv.dat` donde se encuentra el libro. El formato se especifica en la Tabla 6.2. Destaca que la codificación de las páginas viene especificada de forma visible con los caracteres `JPEG` en una de las cabeceras.

Como se ha podido apreciar, una página se compone de diferentes imágenes `JPEG` concatenadas. Tras este análisis, en ninguno de los dos ficheros intermediarios en formato `PSAR` y `GVD` se han encontrado mecanismos que intenten proteger u ofuscar la extracción del libro. Incluso la codificación de las páginas es estándar. Como consecuencia, a pesar de que el libro físico de la versión para Nintendo DS no se llegó a distribuir de forma ilícita en Internet por el trabajo de escanear 300 páginas, el de esta versión sí puede ser encontrado fácilmente.

Cabe destacar que la PS3 provee de mecanismos nativos para cifrar archivos que sí se usan para proteger el resto de datos del juego como imágenes y textos (archivo `hs06_es.adat.sdat`). Este formato con extensión `sdat` y cabecera `NPD` usa las claves y mecanismos internos de cifrado de la consola que asegura su confidencialidad[5].

[3] `http://www.spriters-resource.com/`

[4] `https://github.com/pleonex/Ninokuni/tree/master/Programs/PS3/NinoDecompiler`

[5] A día de hoy ya se ha podido romper este mecanismo. `https://github.com/Hykem/make_npdata`.

Tabla 6.1: Especificación de formato `PSAR`.

Posición	Tamaño	Descripción
		Cabecera
0x00	0x04	Identificador: `PSAR`
0x04	0x04	Versión: `0x00010004`
0x08	0x04	Compresión: `zlib`
0x0C	0x04	Posición de los datos
0x10	0x04	Tamaño de la siguiente sección
0x14	0x04	Número de archivos
0x18	0x04	Reservado
0x1C	0x04	Reservado
		File Info Table
0x00	0x10	Código MD5 del nombre del fichero
0x10	0x04	Desconocido
0x14	0x05	Posición de los datos
0x19	0x05	Tamaño de los datos
		Archivos[a b]

[a] Los archivos pueden estar comprimidos.
[b] El primer archivo con código MD5 nulo contiene la lista de nombres y rutas de ficheros.

<◁>Little Women<▷><:>Louisa May Alcott<○><♀>Chapter 1. <◁>Playing Pilgrims<▷>א<Christmas won't be Christmas without any presents,א= grumbled Jo, lying on the rug.<►><☎>א<It's so dreadful to be poor!א= sighed Meg, looking down at her old dress.<►><☎>א<I don't think it's fair for some girls to have plenty of pretty things, and other girls nothing at all,א= added little Amy, with an injured sniff.<►><☎>א<We've got father and mother and each other,א= said Beth contentedly, from her corner.<►><☎>The four young faces on which the firelight shone brightened at the cheerful words, but darkened again as Jo said

Figura 6.1: Extracto de texto de un libro de *100 Classic Books Collection*.

Conclusión

Puntos fuertes

- El libro está codificado en un formato propietario, se requiere de software no estándar para extraer las páginas.

Puntos débiles

- El archivo no está cifrado, a pesar de que consola provee de mecanismos para ello que sí se utilizan sobre el resto de los ficheros.
- La codificación de las páginas es estándar (`JPEG`).

6.1.2. 100 Classic Book Collection

100 Classic Book Collection fue desarrollado por *HaperCollins* en 2008 para la Nintendo DS. No se trata de un juego, sino de una colección de libros junto a un visor que simula ser un *e-book*. La aplicación contiene en torno a 100 libros además de descargables.

El formato que contiene los libros no ha presentado ningún mecanismo de protección, siendo incluso más sencillo que el caso anterior (Tabla 6.3). De los 23 bloques que contiene un libro, el primero tiene información general, del segundo al quinto imágenes de portada y el resto el texto junto a resumen y descripción del autor (Figura 6.1).

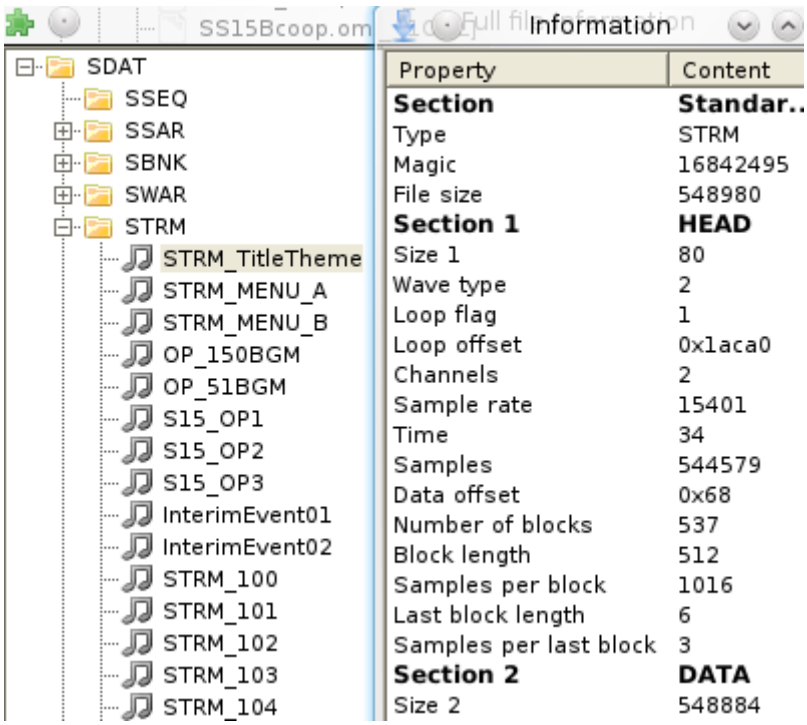


Figura 6.2: Archivos comprimidos en `SDAT` y codificados con IMA-ADPCM visto con Tinke.

A pesar de que las obras son clásicas, sin derechos de autor, debería ofrecer una protección por el trabajo de maquetado, traducción y adaptación.

Conclusión

Puntos fuertes

- No se usa un formato estándar para la representación del texto ni la estructura del libro. Se requiere de programas especializados para extraer y editar.

Puntos débiles

- Los libros no están cifrados.
- No hay mecanismos de integridad para protegerlos ante modificaciones.

6.2. Bandas sonoras

6.2.1. Osu! Tatakae! Ouendan!

Osu! Tatakae! Ouendan! es un juego japonés desarrollado por *iNis* para la Nintendo DS en 2005. En occidente salió una segunda versión que usaba el mismo motor, *Elite Beat Agents*. Son juegos musicales en los que hay que seguir el ritmo de canciones con el lápiz táctil. Internamente los dos juegos son idénticos cambiando el contenido pero no la estructura.

Las canciones están contenidas en un archivo de tipo `SDAT`, estándar en la consola. El formato permite almacenar sonido tanto sintetizado como digitalizado. Para este último se permiten las codificaciones *Pulse-Code Modulation* (PCM) de 8 bits, PCM de 16 bits y *Interactive Multimedia Association - Adaptive Differential PCM* (IMA-ADPCM), siendo esta la que más calidad provee y usada en estos juegos (Figura 6.2).

A pesar de que estos formatos y codificaciones no proveen protecciones, por motivos técnicos, las canciones vienen fragmentadas y no es inmediata su recomposición.

Figura 6.3: *Guitar Grip* necesario para jugar a la saga *Guitar Hero: On Tour*.

Conclusión

Puntos fuertes	Puntos débiles
▪ Se requiere desarrollar software para recomponer las canciones al estar divididas.	▪ Los archivos no están cifrados. ▪ Se usan formatos y codificaciones estándar.

6.2.2. Guitar Hero: On Tour

Guitar Hero: On Tour es una serie de tres videojuegos para la Nintendo DS desarrollados por *Vicarius Visions* en 2008 y 2009. Son juegos musicales para simular tocar canciones con una guitarra, desde los años 70 hasta la actualidad. Se caracterizan por incluir una pieza de hardware extra e imprescindible para el juego denominada *Guitar Grip* (Figura 6.3). Este aparato tiene cuatro botones para tocar acordes de la guitarra y se conecta en la segunda ranura de la consola. Gracias a incluir hardware adicional hace que no sea posible piratear el juego [6].

El primer juego de la saga se ha visto que comprime los archivos en `mainFIGS.gfc` y `mainFIGS.gob`. El primer fichero (Tabla 6.4) tiene la información para acceder a los datos del segundo. Tras analizar los bloques de datos, se pudo ver que estaban comprimidos con un algoritmo cuyo descifrado se implementaba en 1.900 líneas de código.

Esta misma compresión se encontró en el juego *Last Window: El secreto de Cape West* para la Nintendo DS. En realizar un análisis completo se tardaría de media unas 60 horas. Por este motivo, implementar algoritmos tan largos y eficientes es una de las mejores estrategias a la hora de proteger datos. En [25] se revela que se trata de la compresión `zlib`. Estudiar el algoritmo usando este juego hubiese resultado más sencillo, pues hay indicios de que se compiló en modo de depuración y los mensajes de error, aunque no se usaban, estaban presentes, dando pistas sobre el funcionamiento de cada parte.

Tras descomprimirlo se observaron carpetas y archivos con nombres descriptivos de su contenido. El nombre de estas carpetas y el formato de los ficheros coinciden con los presentes en siguientes versiones del juego y se describirán a continuación. Esta edición del juego provee de buenos mecanismos frente distribución ilícita, por la necesidad de hardware adicional, y a la extracción de recursos mediante un algoritmo largo y complejo de descifrar.

Las siguientes ediciones del juego para esta consola fueron *Guitar Hero On Tour: Decades* y *Guitar Hero On Tour: Modern Hits*. En estas versiones, los archivos no están empaquetados con el formato `GOB`, ni protegidos de otra forma. Dentro de los juegos hay dos carpetas, *StrmFIGS* y *TracksFIGS*, las mismas que se encontraban en la primera edición dentro del fichero comprimido.

[6] Más tarde se descubrió un parche [10] para jugar usando los botones de la consola y no necesitar *Guitar Grip*.

En las carpetas se puede encontrar las canciones en formato `Vorbis OGG`. En concreto, por facilidades técnicas, se ha separado cada canción en tres pistas, una con la base de guitarra, otra con la de batería y otra con la voz. Las dos primeras están codificadas en `OGG` y la tercera en `HWAS`. La codificación de este formato es `IMA-ADPCM-Vorbis VOX` [11], con una cabecera de 512 bytes descrita en la Tabla 6.5. Mezclando las tres pistas se obtiene una versión audible pero con ruido.

Conclusión

Puntos fuertes

- En la primera edición se comprimen todos los ficheros.
- Los datos están comprimidos en una compresión de 1.900 instrucciones máquina que dificulta su estudio.
- El formato de algunos ficheros de audio no es estándar y no se puede obtener fácilmente una conversión completamente exacta con programas normales.

Puntos débiles

- La compresión `zlib` se incluye con información de depuración innecesaria que facilita su comprensión.
- En las dos ediciones que siguieron los archivos están sin comprimir.
- La mayoría de las canciones vienen codificadas en el estándar `Vorbix OGG`. Además, se da a conocer este formato mediante la extensión de los ficheros.

6.2.3. Guitar Rock

Guitar Rock es un juego desarrollado por *Gameloft* en el año 2008 para la Nintendo DS. Su jugabilidad es muy parecida a la de *Guitar Hero: On Tour*, diferenciándose en la falta de necesidad de hardware como el *Guitar Grip*.

Las canciones no están protegidas dentro del juego, ya que se encontraron dentro de los archivos *trackX.bar*. Este formato tiene una cabecera de 512 bytes siguiéndole tres pistas musicales de la canción en formato estándar `SWAV`. Mezclando estas tres pistas se obtiene la canción original sin pérdidas.

Conclusión

Puntos fuertes

- Los archivos vienen comprimidos en un formato no estándar.

Puntos débiles

- Las canciones no están cifradas y usan codificaciones estándar.

6.2.4. Level-5

La compañía *Level-5* se centra en el desarrollo de juegos para la Nintendo DS. En este apartado se estudiará la codificación de audio que usa en videojuegos como *El Profesor Layton*, *Inazuma Eleven* y *Ninokuni*.

El formato `SADL` es un contenedor de audio que admite las codificaciones IMA-ADPCM y una desarrollada por el estudio *Procyon*. Esta última se basa en un algoritmo predictivo de coeficientes variables, que ofrece gran calidad de audio. El programa *SADLer* implementa un codificador y decodificador de este formato[7].

En el caso de *Ninokuni* para Play Station 3, se ha usado un formato estándar admitido por los reproductor comunes. La protección en este juego se basa en el contenedor de los archivos, excepto del libro electrónico (Sección 6.1.1), que se cifra con una clave única de la consola.

[7] `https://github.com/pleonex/SADL-Audio-format`

Conclusión

Puntos fuertes

- Se usa un formato y codificación propietaria no documentada.
- En el juego de Play Station 3 los archivos de audio están cifrados con el contenedor `NPD`.

Puntos débiles

- En los juegos de Nintendo DS no se provee de cifrado u otro mecanismo de protección.

6.2.5. Duet

Duet es un juego para las plataformas iOS y Android, desarrollado por *Kumobius* en 2013. La aplicación es de pago pero en periodos de oferta está disponible gratuitamente.

La banda sonora fue compuesta por Tim Shiel y puede comprarse en iTunes[8] por $4.99. Sin embargo, los archivos dentro de la carpeta de la aplicación carecen de seguridad. Si se cuenta con un móvil con acceso al sistema de archivos (*jailbreak* en el caso de iOS) y se accede a su directorio, se puede encontrar toda la banda sonara en formato `WAV`.

Conclusión

Puntos fuertes

- Si no se posee de un dispositivo con acceso al sistema de archivos, no se puede acceder a los audios.

Puntos débiles

- Los ficheros vienen en formato estándar, sin cifrar y con una extensión que indica su contenido y formato.

6.3. Vídeos

Por último lugar, en esta sección se comentará la codificación encontrada en los archivos de vídeo de Play Station 3 y Nintendo DS.

6.3.1. Play Station 3

En esta consola, la protección de muchos archivos se confió en la imposibilidad de descifrar archivos con una clave que sólo se conocía por la consola, como se discutió en el Apartado 6.1.1.

Estos archivos tienen la extensión `pam`, tratándose de una codificación `MPEG` con una cabecera de `0x800` bytes opcionales para su decodificación [1]. El programa *pam2mpg* realiza estas labores de conversión[9].

Conclusión

Puntos fuertes

- La cabecera y extensión no indica la codificación.

Puntos débiles

- Se ha usado una codificación estándar sin cifrado. Sin embargo, se entiende en este tipo de contenido debido a los requisitos técnicos de calidad, compresión y eficiencia.

[8] `https://itunes.apple.com/us/album/duet/id721804015`
[9] `https://github.com/pleonex/Ninokuni/tree/master/Programs/PS3/pam2mpg`

6.3.2. Nintendo DS

En esta consola, la mayoría de juegos usa una codificación propietaria proporcionada por Nintendo y desarrollada por *Nintendo European Research and Development*[10].

Hasta el momento, esta codificación no se ha conseguido documentar debido a su complejidad y a sus características únicas. Es un algoritmo desarrollado específicamente para consola y su hardware. La alternativa a la hora de extraer estos recursos es grabar la pantalla mientras se reproduce el juego en un emulador[11].

Conclusión

Puntos fuertes

- Al usar una codificación única y compleja no se ha podido documentar.

[10] También conocida como *Mobiclip* y *Actimagine*.
[11] El emulador DeSmuME tiene esta funcionalidad implementada.

Tabla 6.2: Especificación de formato `GVD`.

Posición	Tamaño	Descripción
		Cabecera
0x00	0x08	Identificador: `TGDT0100`
0x08	0x04	Número de páginas
0x0C	0x04	Posición de los datos
		Bloques GVD[a]
0x00	0x04	Posición del nombre[b c]
0x04	0x04	Tamaño del nombre
0x08	0x04	Posición de los datos[b]
0x0C	0x04	Tamaño de los datos
		Información de la página
0x00	0x04	Identificador bloque: `GVEW0100`
0x04	0x04	Identificador imagen: `JPEG0100`
0x08	0x04	Ancho de la imagen con más calidad
0x0C	0x04	Alto de la imagen con más calidad
0x10	0x04	Identificador: `BLK_`
0x14	0x04	Tamaño del bloque
0x18	0x04	ID
0x1C	0x04	Reservado
0x20	0x04	Constante `0x20`
0x24	0x04	Constante `0x04`
		Datos de la página[d]
0x00	0x04	Posición X de la imagen en la página
0x04	0x04	Posición Y de la imagen en la página
0x08	0x04	Calidad
0x0C	0x04	Tamaño de los datos
0x10	0x04	Desconocido
0x14	0x04	Desconocido
0x18	0x04	Ancho
0x1C	0x04	Alto
0x20	Variable	Imagen `JPEG`[e]
		GVMP
0x00	0x04	Identificador: `GVMP`
0x04	0x04	Número de bloques.
0x08	0x04	Posición de los datos del primer bloque
0x0C	0x04	Tamaño de los datos del primer bloque

[a] Hay un bloque por página.
[b] Relativo al inicio del bloque de datos.
[c] La codificación del nombre es `ASCII`.
[d] Hay un bloque por cada calidad.
[e] Los datos pueden estar comprimidos en `GVMP`; en ese caso el primer bloque será la imagen `JPEG`.

Tabla 6.3: Especificación del formato de *100 Classic Books*.

Posición	Tamaño	Descripción
	Cabecera	
0x00	0x10	Desconocido
0x10	0x04	Desconocido
0x14	0x04×23	Puntero a cada bloques
0x70	0x04×23	Tamaño de cada bloque
	Bloques[a b]	

[a] Algunos bloques pueden estar vacíos.
[b] Todos los bloques excepto el primero están comprimidos con `LZSS`.

Tabla 6.4: Especificación del formato de *Guitar Hero: On Tour*.

Posición	Tamaño	Descripción
	Cabecera[a b]	
0x00	0x04	Identificador: 0x00008008
0x04	0x04	Tamaño del archivo GOB
0x08	0x04	Número de bloques
0x0C	0x04	Número de archivos
	Tabla de bloques[c]	
0x00	0x04	Tamaño del bloque
0x04	0x04	Posición del bloque
0x08	0x04	Posición del siguiente bloque[d]
0x0C	0x04	Tipo de bloque: 0x30 descomprimido, 0x7A comprimido con *zlib*
	Tabla desconocida[c]	
0x00	0x04	Desconocido
	Tabla de ficheros[c]	
0x00	0x04	CRC32 del nombre en minúscula
0x04	0x04	Tamaño del fichero
0x08	0x04	Posición del primer bloque

[a] Información extraída de ungob.pl [25] y corroborada.
[b] Los datos del fichero están codificados en *big-endian*.
[c] Se describe la primera entrada.
[d] Si es el último bloque el valor será 0x7FFF.

Tabla 6.5: Especificación del formato `HWAS`.

Posición	Tamaño	Descripción
	Cabecera[a b]	
0x00	0x04	Identificador: hwas
0x04	0x04	Desconocido
0x08	0x04	Frecuencia de muestreo
0x0C	0x04	Número de canales
0x10	0x04	Reservado
0x14	0x04	Tamaño de los datos de audio
0x18	0x04	Número de muestras
0x1C	0x1E4	Reservado
	Datos de audio en IMA-ADPCM	

Capítulo 7

Servicios en línea

Este capítulo tiene como objetivo analizar y mostrar los protocolos de comunicación de algunas plataformas y juegos, así como la seguridad aplicada sobre los ficheros transmitidos. Este tipo de contenido se está volviendo más popular gracias a las tiendas virtuales, donde se venden pequeños extras.

En la sección multijugador se explicará cómo funciona la autenticación en los servidores de Nintendo y cómo puede afectar a la jugabilidad no usar `HTTPS` en el caso del juego *Preguntados*. En la sección de contenidos descargables se comentará la seguridad de tres juegos. Por último, la sección de transmisión segura de código explica cómo se implementan mecanismos de integridad para enviar juegos entre consolas.

7.1. Multijugador

7.1.1. Conexión segura en servidores de Nintendo

El 20 de mayo de 2014 Nintendo cesó su servicio de conectividad Wi-Fi para las consolas Nintendo DS y Wii [19]. Los juegos en línea, torneos, contenido descargable e intercambio de objetos se deshabilitaron con el cierre de estos servidores [18]. Por ello, un grupo de usuarios investigó el protocolo de comunicación entre la consola y el servidor, creando una aplicación web que replicase el funcionamiento[1]. En este apartado se estudiará el protocolo.

La Sección 4.3 del capítulo de metodologías explica cómo modificar el emulador DeSmuME de Nintendo DS para capturar tráfico. Tras conseguir una captura (Figura 7.1) se pudo ver que, aparte de una primera comunicación `HTTP`, el resto se cifra mediante `HTTPS`. Para poder eludir la capa de seguridad de `TLS`, se descubrieron dos mecanismos: capturar la función de cifrado y forzar el uso de `HTTP`.

El primero consiste en encontrar la función que cifra la comunicación en el juego, el algoritmo `RC4`. Dado que este código se incluye con la biblioteca que proporciona Nintendo a los desarrolladores, será igual en todos los juegos. Esta táctica es la que implementa el programa *RC4Finder*[2] para encontrar y devolver la posición de la función.

[1] `https://github.com/polaris-/dwc_network_server_emulator`
[2] `https://github.com/pleonex/AiroRom/tree/master/Programs/RC4Finder`

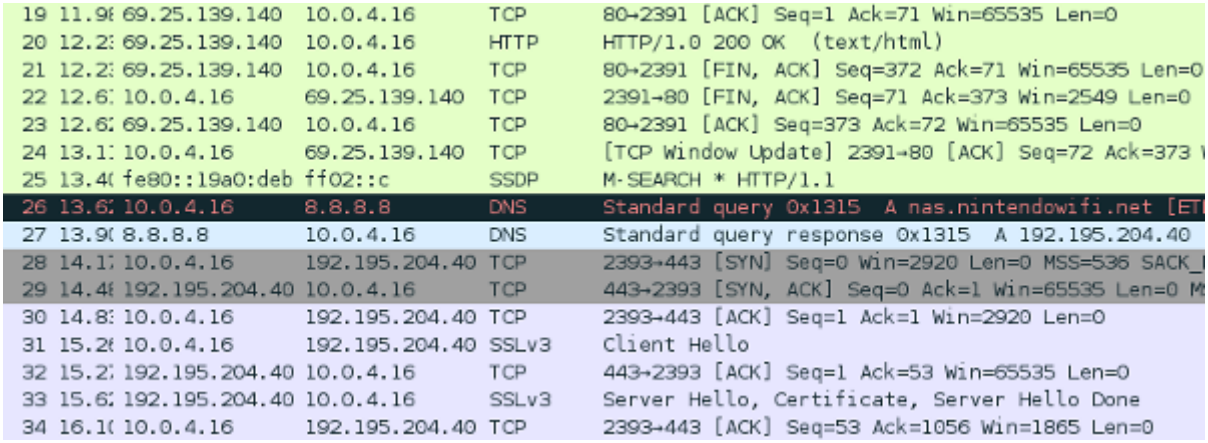


Figura 7.1: Primeros paquetes de una comunicación entre una Nintendo DS y los servidores.

Conociendo dónde se encuentra el algoritmo de cifrado y aprovechando las capacidades de depuración del emulador, se puede implementar una funcionalidad para que guarde en un fichero los datos que pasan por el algoritmo. Dado que el mismo algoritmo se usa para cifrado y descifrado (es un cifrado simétrico), hará falta controlar dos puntos de la función. Si se quieren obtener los datos que envía el juego, serán aquellos que se van a cifrar y que, por tanto, será el contenido inicial al principio de la función. En el caso de querer guardar los datos que se reciben, los que se descifrarían, que estarán al final del algoritmo. El mecanismo se ha implementado usando el método `HandleDebugEvent_Execute` del archivo `debug.cpp`. Este se llama después de haber procesado una instrucción, por lo que se puede utilizar para realizar una acción al ejecutarse una parte concreta del juego. En el repositorio del proyecto se encuentra el archivo con la implementación[3]. Hay que tener en cuenta que necesita ser modificada para cada juego para poner el inicio y el final del algoritmo `RC4`.

El segundo mecanismo se debe a un fallo de seguridad por parte de Nintendo. Los servidores están mal configurados y **admiten tanto peticiones `HTTPS` como `HTTP`**, ofreciendo la misma funcionalidad. Además, con solo cambiar la parte de protocolo de la `URL` en el código del juego, se adapta automáticamente y deja de cifrar la comunicación. El procedimiento de editar todas las direcciones para usar una conexión sin cifrar se ha implementado en el programa *SSLPatcher* incluido en el repositorio del trabajo[4]. Solo los servidores de contenido descargable denegaban conexiones al puerto 80 (el usado en `HTTP`).

Una vez capturados los paquetes sin cifrar (Figura 7.3), se pudo analizar la comunicación entre un total de tres servidores. En el diagrama de la Figura 7.2 se describen los mensajes con los parámetros más importantes:

1. Prueba de conexión. Se realiza una prueba de conectividad con una petición `HTTP` a la dirección `conntest.nintendowifi.net`. Este devuelve una página `HTML` con el texto *This is a test.html page.*

2. Autenticación en servidor *Network Access Server* (NAS). Primera conexión de autenticación con el servidor NAS. El juego se autentica mediante el comando `login`, pasando como parámetros un identificador del juego (`gamecd`) y una clave (`passwd`). Adicionalmente, la consola también envía un ID de usuario, el ID del desarrollador del juego, el BSSID del punto de acceso, la dirección MAC de la consola, el lenguaje configurado, la fecha de cumpleaños del usuario y el nombre del usuario. El servidor contesta enviando un *token* generado y un *challenge* que se explicarán más tarde.

[3] `https://github.com/pleonex/AiroRom/blob/master/Programs/DeSmuMEPCAP/debug.cpp`
[4] `https://github.com/pleonex/AiroRom/tree/master/Programs/SslPatcher`

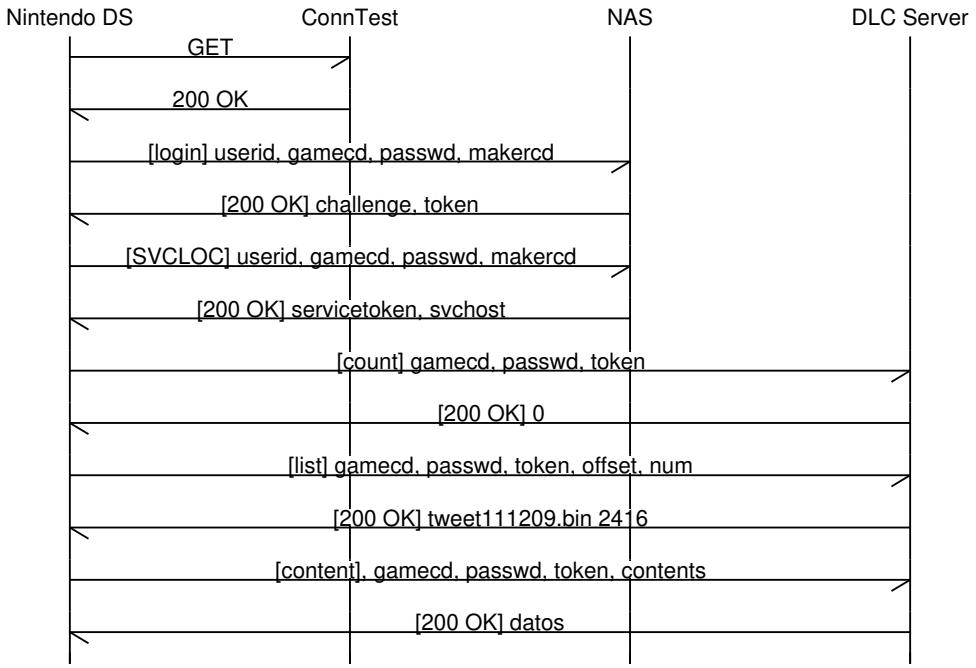


Figura 7.2: Comunicación entre Nintendo DS y servidor de Nintendo para descargar un fichero.

3. Localización de servidor externo. Esta conexión se realiza cuando se pide contactar con un servidor externo como de contenidos descargables (comando `svcloc`). Se descartan los datos de la operación anterior (*token* y *challenge*) y se devuelve uno nuevo para ese servicio en concreto (`servicetoken`) junto con la dirección a la que contactar (`svchost`).
4. Contacto con el servidor de contenidos. En primer lugar el juego envía siempre el comando `count` donde opcionalmente puede especificar un filtro. El servidor devolvería el número de ficheros que coinciden con dicho filtro. Como en este caso no se ha especificado ninguno, devuelve el valor cero. Para autenticarse en este servidor usa tanto el `servicetoken` que devolvió el NAS como el código del juego, y una contraseña que es diferente a la usada en el servidor anterior.
5. Obtención del nombre de los ficheros. El juego envía el comando `list` para obtener la lista de ficheros disponibles. Para realizar un filtro sobre esa lista, utiliza los campos `offset` (índice de la lista por el que comenzar) y `num` (número de elementos a devolver). El servidor envía en cada línea el nombre del fichero y su tamaño.
6. Finalmente, el juego pide un fichero mediante el comando `content`.

Este protocolo tiene ciertos **fallos tanto de seguridad como de eficiencia**:

- En el caso de querer contactar con el servidor de descargas (caso descrito anteriormente), la primera comunicación con el NAS es innecesaria.
- La contraseña enviada al servidor NAS no se verifica si coincide con el código de juego, pudiendo especificar la de otro y dejarla constante. Sin embargo, la que se usa en el servidor de descargas sí se comprueba.
- La autenticación con el servidor de contenidos es de tipo *Password Authentication Protocol* (PAP) (solo se utiliza una contraseña), frente la del servidor de juegos en línea que es *Challenge Handshake Authentication Protocol* (CHAP) (autenticación mediante *tokens*).
- Conociendo la contraseña de un juego y gracias al comando `list`, se puede crear un programa que solicite todos los ficheros del servidor y los descargue. Esto se hizo a la hora de recuperar los contenidos descargables cuando se conoció que los servidores de Nintendo se iban a desactivar.

```
POST /ac HTTP/1.0
Content-type: application/x-www-form-urlencoded
Host: nas.nintendowifi.net
User-Agent: Nitro WiFi SDK/5.3
HTTP_X_GAMECD: B2KJ
Connection: close
Content-Length: 274

action=bG9naW4*&gsbrcd=&sdkver=MDA1MDAz&userid=Nzg4MTk4NjAyOTYzMQ**&passwd=Mjk3&bssid=MDB
mMDFhMmIzYzRk&apinfo=MDI6MDAwMDAwMCOwMA**&gamecd=QjJLSg**&makercd=SEY*&unitcd=MA**&macadr
=MDAxNjU2ODMzMWY5&lang=MDU*&birth=MDcwYw**&devtime=MTQwNDI5MTMxMDA1&devname=UABhAFIAYQBEA
G8AWAA*HTTP/1.1 200 OK
NODE: wifiappw3
Content-Type: text/plain
Content-Length: 235
Date: Tue, 29 Apr 2014 11:10:14 GMT
Connection: close
Server: Nintendo Wii (http)

challenge=WUpQUVhOOVQ*&locator=Z2FtZXNweS5jb20*&retry=MA**&returncd=MDAx&token=TkRTTExGMW
pzQnRyMmVONTcyMjVnSmZPY2V0cHozK0xmbVJ4Y1gyczRKNmJQWG90aWdhK3Q2bUVLeUNaZ0kyWFovYzVKT2QwZlp
ScS9rSFRDV2phd1lOanc9PQ**&datetime=MjAxNDA0MjkxMTEwMTU*
```

Figura 7.3: Petición respuesta descifrada entre Nintendo DS y servidor.

```
\lc\1\challenge\QZVIOS5LDX\id\1\final\\login\\challenge\iA5bLRk9tEJJTAq4SgZYrgKSeHiCXcN0
\authtoken\ND5pLfvzD++lkhqnAc1eAHDr+yYAZqIn7QsvrkcGawYuskg+2cHa4EpMI998SwD90Vedt0L5uA
+owew3TFq03QN5vuOTStPBa8F8xRjUrFaYsQwO9OvGl5beR7WuKTN1qNX\response
\ef5f198e7ca7d50d238cea2fff707f82\firewall\1\port\0\productid\10854\gamename\wormsow2ds
\namespaceid\0\id\1\final\\lc\2\sesskey\9018710\proof\0f4e707f1dd0ad79989feb7f418aaa1f
\userid\448374481\profileid\480588010\uniquenick\75clfke1vAW2E0hg9bf0\lt\xKPwlfsjgGchmzN
[dg4yMQ__\id\1\final\\getprofile\\sesskey\9018710\profileid\480588010\id\2\final\\pi\
\profileid\480588010\nick\75clfke1vAW2E0hg9bf0\userid\448374481\email
\75clfke1vAW2E0hg9bf0@nds\sig\8cd617791e3dc8de51fc3d80c20f38b8\uniquenick
\75clfke1vAW2E0hg9bf0\pid\11\lon\0.000000\lat\0.000000\loc\\id\2\final\\getprofile\
\sesskey\9018710\profileid\480588010\id\3\final\\updatepro\\sesskey\9018710\lastname
\0efk6169dAW2E3o9g260\final\\pi\\profileid\480588010\nick\75clfke1vAW2E0hg9bf0\userid
\448374481\email\75clfke1vAW2E0hg9bf0@nds\sig\8cd617791e3dc8de51fc3d80c20f38b8\uniquenick
\75clfke1vAW2E0hg9bf0\pid\11\lon\0.000000\lat\0.000000\loc\\id\3\final\\ka\\final\
```

Figura 7.4: Mensajes descifrados del protocolo multijugador de Nintendo DS. En verde el *token*.

Respecto a la comunicación `HTTPS`, la implementación sobre la consola es segura y no se pueden usar servidores alternativos sin modificar el juego. Dentro de este se encuentran los certificados que los servidores de Nintendo utilizan durante el intercambio de claves del protocolo `HTTPS`. El juego comprueba que el certificado no haya expirado, no tenga problemas de formato, que la firma sea válida y que coincida con uno de los de su lista. Sin embargo, comprueba el campo del nombre del servidor. Por defecto, los juegos incluyen un certificado a nombre de Nintendo, dos de VeriSign, tres de CyberTrust, dos de Thawte y dos de GlobalSign.

Para finalizar, se investigó el algoritmo usado para generar un *token* de autenticación en los juegos en línea (Figura 7.4). En ese caso, después de la primera conexión con el NAS, se inicia una comunicación con el servidor multijugador cuyo protocolo está basado en *GameSpy*. Para contactar, es necesario que los *tokens* generados por la consola y servidor coincidan. Iniciada la conexión `TCP`, este le enviará un primer mensaje con un segundo *challenge*. El *token* que se calcula es el resultado de aplicar el algoritmo `MD5` sobre la cadena de caracteres formada al concatenar los siguientes parámetros (conocidos por ambas entidades).

- `MD5` del *challenge* del servidor NAS.
- 48 espacios en blanco
- *Token* del servidor NAS
- *Challenge* generado por la consola que se transmitirá junto a este mensaje.
- *Challenge* enviado por este servidor.
- `MD5` del *challenge* del servidor NAS.

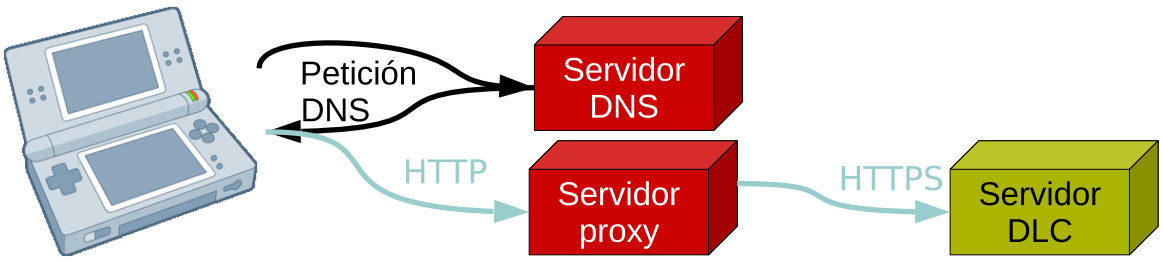


Figura 7.5: Conexión a los servidores alternativos de Nintendo.

{'id':4080,"category":"HISTORY","text":"..C..mo se denominaba en la Edad Media al hijo
tenido fuera del matrimonio?","answers":
["Fogardo","Bastardo","Hu..rfano","Primog..nito"],"correct_answer":1,"media_type":"NORMAL
"}},{"question":{"id":4004,"category":"GEOGRAPHY","text":"..Qu.. colores tiene la bandera
de M..xico?","answers":["Azul, amarillo y rojo","Azul, verde y rojo","Blanco, verde y
rojo","Amarillo, verde y
rojo"],"correct_answer":2,"media_type":"NORMAL"},"powerup_question":

Figura 7.6: Captura de tráfico con las preguntas y respuestas de una partida de *Preguntados*.

De esta forma solo aquellos dispositivos que reúnan tanto los códigos devueltos por el servidor que da acceso a la red como el algoritmo para calcularlo serán capaces de comunicarse con éxito. Esta parte fue clave a la hora de realizar un servidor alternativo.

Dado que no se conocen las claves privadas de Nintendo, no se puede realizar una conexión segura con el juego y el servidor alternativo. Para solventar el problema sin tener que modificar los certificados existentes se usa un servidor *proxy* de `DNS`, como se explica en el diagrama de la Figura 7.5. Este servidor DNS resuelve los dominios de Nintendo apuntando a los alternativos.

7.1.2. Preguntados

Preguntados es un juego desarrollado por *Etermax* en el año *2013* para las plataformas iOS, Android y Windows Phone. La aplicación encuentra adversarios de forma aleatoria y muestra una serie de preguntas con posibles opciones.

El objeto de este estudio es comprobar si la seguridad en las comunicaciones es segura, evitando posibles trampas. Los paquetes capturados con *Wireshark* usando una metodología similar a la descrita en la Sección 4.3, muestran que no se usa una conexión segura, ya que se transmite todo por `HTTP`. Como se ve en la Figura 7.6, tras analizar la transmisión de datos con el servidor del juego, se concluye una mala implementación del protocolo. Cada vez que una partida comienza, el servidor le envía al usuario una lista de preguntas y respuestas a usar **junto a la respuesta correcta**. Dado que la comunicación no está protegida y se puede capturar, es posible saber la respuesta a cualquier pregunta antes de que se realice. Aparte de los problemas de seguridad y confidencialidad que esto implica, esto permitiría crear un programa que responda automáticamente a las preguntas.

Una posible solución aparte de usar `HTTPS` sería que el servidor enviase solo las preguntas con sus posibles respuestas, y que la respuesta escogida se comunique al servidor donde se comprobaría.

7.2. Contenido descargable

7.2.1. 100 Classic Books Collection

100 Classic Books Collection se trata de un juego que, como se ha visto en la Sección 6.1.2, ofrece al usuario un lector de libros electrónicos. Este juego provee además de una opción de comunicación inalámbrica para descargar alrededor de 20 nuevos libros. Estos se almacenarían en el archivo de guardado del usuario. Tras analizar las comunicaciones, usando la versión modificada de *DeSmuME*, se pudo observar que estos ficheros descargados no estaban cifrados, teniendo el mismo formato que aquellos que se encuentran dentro del juego.

306	TFPremiumUnlocked	0	12	1
307	TFEncoreUnlocked	0	12	1

Figura 7.7: Filas de la base de datos de *Duet* que activan el contenido extra.

Reproducir contenido desde un archivo de guardado, teniendo en cuenta que se añade mediante descarga externa, es una vulnerabilidad a explotar para ejecutar código no autorizado. Se podría encontrar un fallo de seguridad como un desbordamiento de *buffer* al insertar un texto muy largo y obtener el control de la consola. Esta es la técnica que se utiliza para ejecutar copias de seguridad de juegos en Nintendo DSi.

7.2.2. Ninokuni: El Mago de las Tinieblas

Ninokuni es un juego que, como se comentó en la Sección 5.2, es para la *Nintendo DS*. *Level-5* ofrecía dos servicios adicionales para el juego mediante la conectividad inalámbrica. El primero se trataba de una lista de noticias cortas relacionadas con la historia que salían una vez al día, y la segunda era contenido adicional como objetos o misiones. Ambas se obtenían descargando unos archivos de los servidores de Nintendo con el protocolo descrito en el Apartado 7.1.1.

Usando la versión modificada de DeSmuME se pudo obtener los paquetes descifrados, pero no se encontró texto dentro de los binarios que se descargaban. Tras investigar las instrucciones máquina que procesaban, se vio que se usaba el algoritmo `RC4` para descifrar los archivos. Por tanto, en una comunicación normal, este algoritmo se usaba dos veces, una para la capa de sesión `TLS` y otra en la capa de aplicación. Las claves en esos dos casos son diferentes, pues en la última es una constante almacenada en el juego mientras en la primera se crea para cada conexión.

Aparte del cifrado, se encontró un algoritmo de integridad en la posición `0x04` de ambos ficheros. Concretamente se aplica el algoritmo `CRC32` que, aunque no es necesario porque se usa `TCP`, asegura que el juego no procese datos no esperados.

7.2.3. Duet

Por último se verá el caso del juego *Duet* para *iOS* y sus niveles extras. Se trata de una compra integrada en la aplicación por 0,99 euros que añade un nuevo conjunto de pruebas. Analizando el contenido de los documentos del juego, se pudo encontrar indicios de que estos niveles estaban presentes antes de realizar la compra, por lo que simplemente se activarían una vez el pago hubiese finalizado. La búsqueda se centró en saber dónde se guardaba la configuración del juego para ver si estaba protegida, de forma que no se pudiese evitar la compra activándolos manualmente.

En la carpeta *Documents* de la aplicación existen tres bases de datos *sqlite*. Abriendo la de mayor tamaño, *persistent-data.db*, nos encontramos las filas de la Figura 7.7. En ella se ve cómo los valores que activan el contenido extra, el contenido de pago, están puestos a `0`, el valor que corresponde a *desactivado*. Si se cambia a `1` y se inicia la aplicación, de nuevo nos encontraremos con este contenido activado.

La protección ante este tipo de casos es tan sencilla como poner una contraseña a la base de datos. Se trata de un mecanismo que está implementado nativamente[5] en las bibliotecas de *sqlite* y es muy sencillo de usar. La contraseña se puede almacenar en texto plano en la aplicación, pues dificultará la tarea de conocerla y, al valer el contenido tan poco (menos de 1 euro), no compensará el esfuerzo dedicado como se ha discutido en capítulos anteriores.

[5] `http://stackoverflow.com/a/24349415/3021815`

7.3. Transmisión segura de código

Una de las posibilidades de comunicación inalámbrica que ofrece la NDS es que un juego envíe una *demo* a otra consola, en algunos casos incluso para poder jugar en multijugador teniendo un único juego, denominado *Download Play*.

En este contexto, dado que la transmisión de código, se hace en un canal compartido (se usa el protocolo *Wi-Fi*), son necesarios mecanismos que protejan la comunicación. De esta forma, no sería posible realizar un ataque *man-in-the-midle* para ejecutar código no autorizado introduciendo así una brecha de seguridad.

El protocolo [15] implementa varios mensaje, siendo los más importantes en cuanto seguridad `RSA signature` y `data`. Con el primero se envían los archivos binarios de código principales de la aplicación, uno por procesador. Este mensaje está además firmado usando `SHA-1` para que no se pueda alterar el código. El segundo mensaje sirve para transmitir el resto de datos del juego como imágenes, textos y sonidos. Sin embargo, en algunos casos, dado que el fichero principal de código es muy grande, este se divide en partes llamados *overlays*. Estos archivos se tratan como ficheros normales y son transmitidos en el segundo paquete, que no está firmado.

Para solventar este problema, Nintendo introdujo una comprobación de seguridad que, a diferencia de la que se aplica sobre el archivo principal, se realiza durante la ejecución del juego. A cada archivo de código secundario se le realiza un algoritmo de tipo `HMAC` (`SHA-1` para el *hash*), con una clave que se guarda en el archivo de código principal. Este resultado se comprueba con uno almacenado y en caso de fallar se detiene la ejecución del juego.

Este procedimiento, aunque implementado en la mayoría de juegos, solo se activa si se está ejecutando con mediante *Download Play*. En este modo, se activa un bit del *firmware* (posición en la memoria RAM `0x27FFC40`), y durante la ejecución se comprueba para saber si hay que verificar los archivos. En otro caso, como el de un inicio normal desde el cartucho, se asume que no es necesario pues no han sufrido modificación externa.

Ese valor está desactivado en emuladores como DeSmuME y *flashcards* para poder aplicar parches antipiratería y saltarse el mecanismo. De esta forma, una vez modificado el juego original, se puede enviar a otras consolas el juego editado, permitiendo ejecutar código no seguro.

Capítulo 8

Resultados y recomendaciones

A lo largo de la memoria se han analizado una serie de juegos y los algoritmos adaptados de cara a la provisión de seguridad. En esta sección se resumirán los más importantes y se propondrán mejoras en algunos casos.

Es de señalar la relevancia de estas recomendaciones, puesto que hasta la fecha no existe reportado nada similar en la literatura.

8.1. Seguridad sobre ficheros

Los Capítulos 5 y 6 muestran los procedimientos de seguridad aplicados sobre ficheros para evitar su modificación y distribución. A continuación, se enumeran los algoritmos por tipo de formato y se proponen soluciones para mejorar.

Comenzando por el acceso a ficheros, se han observado dos tipos de protecciones. La primera consiste en comprimir todos los archivos en uno, con un formato propietario. De esta forma, utilizando programas genéricos de exploración de juegos no se pueden extraer los recursos. Esto se ha visto implementando en el juego *El Profesor Layton y la Llamada del Espectro*[1] y *Guitar Hero: On Tours* (Sección 6.2.2). En este caso, al utilizar la implementación de `zlib` para comprimir los datos, el algoritmo visto en ensamblador ocupaba 1.900 líneas de código y hacía inviable su investigación. La segunda técnica consiste en ofuscar el nombre y clasificación de los ficheros como se vio en *Pokémon Blanco y Negro* (Sección 5.1.3).

En cuanto a archivos con contenido de texto, se ha observado que el procedimiento habitual es realizar un cifrado sobre cada carácter mediante una clave que cambia (juegos de *Pokémon* 5.1). Esto provee de un buen mecanismo a nivel de eficiencia y ofuscación, requiriendo conocimientos de depuración para encontrar la clave. El principal problema es usar la operación `XOR` con una clave estática en un fichero que tiene varios bytes nulos seguidos, ya que se estaría guardando la contraseña en texto plano dentro del fichero cifrado (caso de *Pokémon Perla* en los campos de *posición* y *tamaño*, Apartado 5.1.1 y *Ninokuni* 5.2). La solución es usar una clave variable, por ejemplo como se hace en los juegos de Pokémon, sumando una constante a aquella.

Además, usar una codificación no estándar para los caracteres dificulta la depuración, ya que no se puede encontrar texto ni en el archivo binario del juego ni en la memoria RAM. En este caso, para evitar poder usar técnicas de búsqueda diferencial (programa *RelativeSearch*), se deben desordenar los caracteres mezclando en un mismo rango símbolos y letras. Finalmente, para evitar que la nueva codificación se pueda obtener a través de la fuente, se debería cifrar este fichero también.

[1] `https://github.com/pleonex/airorom/wiki/Professor-Layton-and-The-Last-Specter`

Respecto a las imágenes se encontró en *Pokémon Perla y Diamante* un cifrado basado en la operación `XOR` con una clave dinámica (Sección 5.1.1). A pesar de cifrar un archivo es un buen método para proteger su contenido, en el caso de archivos multimedia, debido a la complejidad de implementación para tratar estos contenidos, es mejor crear un nuevo formato. De esta forma, si se hubiese utilizado otra codificación para los píxeles como representar un color en otro espacio o usando un número variable de bits, se hubiese necesitado crear nuevos programas para procesarlos. Esta tarea es más compleja que realizar un bucle que descifre.

Finalmente, tras analizar los archivos con contenido de derechos de autor como libros digitales (Sección 6.1), música (Sección 6.2) y vídeos (Sección 6.3), no se encontraron mecanismos específicos de protección. Aparte de usar codificaciones propietarias como en el caso del sonido `SADL` (Sección 6.2.4) y vídeos de Nintendo DS, los ficheros no estaban cifrados, ni incluían mecanismos de integridad. Estos podrían cifrarse y ofuscarse usando las técnicas anteriormente descritas.

8.2. Seguridad en comunicaciones

Los protocolos de comunicación y seguridad implementada sobe los archivos descargados se analizó en el Capítulo 7.

Respecto al protocolo de Nintendo para juegos en línea en Nintendo DS (Sección 7.1.1), se ofrecen mecanismos de autenticación usando CHAP y cifrando la comunicación en `HTTPS`. Sin embargo, una mala configuración de los servidores permitía que, cambiando la `URL` de acceso en el juego, se pudiese usar `HTTP`, quitando la capa de cifrado. En cuanto al acceso a los servidores de contenido descargable, estos tenían cerrado el puerto 80, bloqueando un acceso no cifrado. Sin embargo, la forma de autenticación es mediante una contraseña incluida en texto plano en el juego (mecanismo PAP). Los ficheros descargados de estos servidores, en el caso de *Ninokuni* (Sección 7.2.2), estaban cifrados mediante `RC4` e incluían integridad con `CRC32`. No pasaba lo mismo con *100 Classic Books Collection* (Sección 7.2.1), donde no se aplicaba ninguna protección al descargar libros digitales.

Se analizaron también dos juegos de plataformas móviles. En *Preguntados* (Sección 7.1.2) se comprobó que, al no usar un protocolo seguro de comunicación, se podía hacer interceptación mediante una configuración *man-in-the-middle*, obteniendo las respuestas correctas a las preguntadas planteadas en el juego. En dicho apartado se proponía tanto cifrar la comunicación como cambiar el diseño del protocolo para no enviar las soluciones al cliente. En el caso de *Duet* (Sección 7.2.3), el contenido descargable de pago se podía activar manualmente cambiando un valor de una base de datos. Cifrando la base de datos, o incluso comprobando que el usuario había realizado dicha compra con un servidor externo, se podría evitar.

Finalmente, se estudió el caso de transmisión de código ejecutable en las demos para Nintendo DS (Sección 7.3). El problema se solucionaba firmando digitalmente el archivo con el código principal durante el envío, y realizando el algoritmo `HMAC` sobre los archivos secundarios de código. Esta configuración evitaba ejecutar código alterado durante un ataque *man-in-the-middle*. Sin embargo, si se modificaba el juego de original, en la segunda consola se podía ejecutar código alterado también. Esto se podría evitar firmando digitalmente el binario principal durante la compilación del juego, en lugar de solo durante el envío inalámbrico. De esta forma se evitaría poder ejecutar código no autorizado[2].

[2]Esto se realiza en la consola Nintendo DSi.

Capítulo 9

Conclusiones

A lo largo del trabajo se han analizado una serie de juegos, estudiando los mecanismos de protección que implementan y señalando sus carencias y debilidades. Todos los objetivos propuestos han sido alcanzados, estudiando un total de 21 juegos. De estos, se han mencionado en esta memoria un total de 14 y documentado el resto en la *wiki* del repositorio del trabajo[1].

Se planteó un objetivo opcional que, por falta de tiempo, no se ha podido cumplir. Este fue crear un depurador de código para Nintendo DS llamado *NitroDebugger*[2], del cual se ha terminado su núcleo y faltaría crear una interfaz gráfica y un programa desensamblador. Este proyecto se presentó al *Certamen de Proyectos Libres de la UGR 2014*. A pesar de no estar entre los finalistas debido al estado de completitud, recibió buenas críticas del jurado[3].

En cuanto a objetivos académicos, se han alcanzado los siguientes:

- Identificar problemas no tratados en la literatura.
- Organizar un trabajo en tareas e investigarlas.
- Desarrollar software adicional en los lenguajes `C#` y `python` para complementar la investigación.
- Utilizar programa de control de versiones (`git`).
- Aprender y usar conceptos de bajo nivel de software y hardware, estudiando protocolos y componentes de la Nintendo DS, así como su lenguaje ensamblador, `ARM`.
- Diseñar metodologías de ingeniería inversa para analizar videojuegos.
- Diseñar estrategias para estudiar las comunicaciones inalámbricas, como han sido modificar el emulador DeSmuME para guardar los paquetes y realizar un diseño *man-in-the-middle* para capturar el tráfico de las aplicaciones móviles.
- Aprender el lenguaje LaTeXpara la redacción de esta memoria.

9.1. Trabajo futuro

Los estudios de este trabajo han pretendido resumir los mecanismos más frecuentes encontrados en los juegos de NDS así como señalar las carencias en plataformas móviles. A continuación se ofrece una lista sobre tópicos en los que los que se podría profundizar:

- Estudiar los mecanismos de seguridad implementados en las videoconsolas.
- Estudiar los mecanismos anti-copia implementados física y digitalmente sobre los videojuegos y aplicaciones móviles.

[1] `https://github.com/pleonex/airorom/wiki/Mecanismos-a-investigar`
[2] `https://github.com/pleonex/NitroDebugger`
[3] `https://goo.gl/g8xdWZ`

- Desarrollar un explorador de juegos avanzado, pudiendo detectar algoritmos y formatos ya estudiados.
- Terminar la implementación del depurador de código remoto.
- Analizar videojuegos de la nueva generación de consolas: Nintendo 3DS, Wii U, Play Station 4 y Xbox One.
- Realizar este estudio sobre aplicaciones de ordenador.
- Estudiar los algoritmos DRM que aplican plataformas como *Steam*.
- Implementar algoritmos propuestos en videojuegos y dispositivos móviles.
- Estudiar protocolos usados por videojuegos de aplicaciones móviles para realizar micropagos.
- Estudiar los *exploits* que las *flashcard* para Nintendo DS, Nintendo DSi y Nintendo 3DS utilizan.
- Estudiar la integridad en los archivos de guardado (relacionado con el punto anterior).

Bibliografía

[1] AlphaTwentyThree. *PS3 .pam video files*. 2010. URL: http://forum.xentax.com/viewtopic.php?p=43489&sid=4381fd6ac4017497844e195c912aa329#p43489 (visitado 28-06-2015).

[2] apassingremark. *DQ7 3DS Localization: That's all folks!* 2015. URL: https://www.reddit.com/r/JRPG/comments/3as3mn/dq7_3ds_localization_thats_all_folks/ (visitado 23-06-2015).

[3] Arxan. *State of Application Security*. 2015. URL: https://www.arxan.com/wp-content/uploads/2015/06/2015-State-of-Application-Security_Vol_4_Infographic.pdf (visitado 02-07-2015).

[4] Entertainment Software Association. *Games: Improving the Economy*. 2014. URL: http://www.theesa.com/wp-content/uploads/2014/11/Games_Economy-11-4-14.pdf (visitado 01-07-2015).

[5] Emma Boyes. «UK paper names top game franchises». En: *GameSpot* (10 de ene. de 2007). URL: http://www.gamespot.com/articles/uk-paper-names-top-game-franchises/1100-6164012/.

[6] The REDO compendium. *Reverse Engineering for Software Maintenance*. inglés. 1993.

[7] Eldad Eilam. *Reversing. Secrets of Reverse Engineering*. inglés. 2005.

[8] Asociación Española de Empresas Productoras y Desarrolladoras de Videojuegos y Software de Entretenimiento. *La industria española del videojuego creció un 21 en 2014*. 2014. URL: http://www.dev.org.es/es/noticias-a-eventos/notas-de-prensa-dev/211-la-industria-espanola-del-videojuego-crecio-un-21--en-2014 (visitado 01-07-2015).

[9] Universidad Politécnica de Cataluña Facultad de Informática de Barcelona. *Historia de los videojuegos*. 2008. URL: http://www.fib.upc.edu/retro-informatica/historia/videojocs.html (visitado 01-07-2015).

[10] FAST6191. *How to change buttons on Guitar Hero: On Tours hack*. 2011. URL: https://gbatemp.net/threads/how-to-change-buttons-on-guitar-hero-on-tour-hack.275789/%5C#post-3411881 (visitado 18-06-2015).

[11] GameZelda. *Guitar Hero On Tour *.hwas codec*. 2008. URL: http://forum.xentax.com/viewtopic.php?p=25994&sid=107c2dd6cbc8adc41b01de7d23584700#p25994 (visitado 18-06-2015).

[12] GBATemp. *Translation Index Thread*. 2015. URL: http://gbatemp.net/threads/translation-index-thread.193740 (visitado 22-06-2015).

[13] Andrew Huang. *Hacking the Xbox. An Introduction to Reverse Engineering*. inglés. 2003.

[14] jduncanator. *How does iOS app DRM work, exactly?* 2012. URL: http://apple.stackexchange.com/a/48236 (visitado 18-06-2015).

[15] Martin Korth. *GBA/NDS Technical Info*. 2014. URL: http://problemkaputt.de/gbatek.htm (visitado 29-06-2015).

[16] Lapo F. Mori. «Writing a thesis with LaTeX». inglés. 2 de dic. de 2008. URL: https://tug.org/pracjourn/2008-1/mori/mori.pdf (visitado 02-07-2015).

[17] James Newton. «"Too Many Hurdles"to Ni No Kuni DS Translation». En: *NintendoLife* (18 de abr. de 2012). URL: http://www.nintendolife.com/news/2012/04/too_many_hurdles_to_ni_no_kuni_ds_translation.

[18] Nintendo. *Información sobre las funciones en línea de Nintendo DS y Nintendo DS Lite*. 2014. URL: https://www.nintendo.com/consumer/wfc/es_na/ds/index.jsp (visitado 28-06-2015).

[19] Nintendo. *Nintendo Wi-Fi Connection service for Nintendo DS and Wii has ended*. 2014. URL: http://www.nintendo.com/whatsnew/detail/vyWpoM6CBIe6FjW8NIY7bvzOrgBURhzw (visitado 28-06-2015).

[20] Bryan Norris y Michelle Humphries. *Digital Rights Management in games*. University of North Carolina-Chapel Hill School of Law. 2013. URL: http://drm.web.unc.edu/games (visitado 18-06-2015).

[21] Stephany Nunneley. *Square issues cease and desist to fans working on Final Fantasy Type-0 translation patch*. 2014. URL: http://www.vg247.com/2014/07/18/final-fantasy-type-0-translation-psp-vita/ (visitado 23-06-2015).

[22] Aline Robert. «Cultural industries unite against copyright reform». En: *EurActiv* (16 de mar. de 2015). URL: http://www.euractiv.com/sections/infosociety/cultural-industries-unite-against-copyright-reform-312894.

[23] RomHacking.net. *Abandoned Projects*. 2015. URL: http://www.romhacking.net/abandoned (visitado 22-06-2015).

[24] John Szczepaniak. *Fan translations*. 2011. URL: http://www.hardcoregaming101.net/Fantranslation/Romhacking.htm (visitado 26-06-2015).

[25] tma. *SlowHero: GOB extractor*. 2008. URL: http://slowhero.moto-coda.org/tech/ungob.pl (visitado 18-06-2015).

[26] Wikipedia. *Always-on DRM*. 2015. URL: https://en.wikipedia.org/wiki/Always-on_DRM (visitado 18-06-2015).

[27] Wikipedia. *Fan translation of video games*. 2015. URL: https://en.wikipedia.org/wiki/Fan_translation_of_video_games (visitado 24-06-2015).

[28] Wikipedia. *Gestión digital de derechos*. 2015. URL: http://es.wikipedia.org/wiki/Gesti%C3%B3n_digital_de_derechos (visitado 18-06-2015).

[29] Wikipedia. *Mod (videojuegos)*. 2015. URL: http://es.wikipedia.org/wiki/Mod_%28videojuegos%29 (visitado 24-06-2015).

[30] Xiao Zhang. *A Survey of Digital Rights Management Technologies*. 2011. URL: http://www.cse.wustl.edu/~jain/cse571-11/ftp/drm/#sec2.4 (visitado 28-06-2015).

Acrónimos

DRM Digital Resource Management

EFF Electronic Frontier Foundation

PS4 Play Station 4

PS3 Play Station 3

PS2 Play Station 2

PSX Play Station 1

PSP Play Station Portable

NDS Nintendo DS

N3DS Nintendo 3DS

DSi Nintendo DSi

GBA GameBoy Advance

XOne Xbox One

GC Game Cube

DC Dreamcast

PCM Pulse-Code Modulation

IMA-ADPCM Interactive Multimedia Association - Adaptive Differential PCM

VWT Variable Width Table

TOC Table Of Content

NAS Network Access Server

PAP Password Authentication Protocol

CHAP Challenge Handshake Authentication Protocol

MIT Massachusetts Institute of Technology

ESPRIT European Strategic Program on Research in Information Technology

DMCA Digital Millennium Copyright Act