UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

Ingeniería de Telecomunicación

COMUNICACIONES SEGURAS EN TIEMPO DE PRE-BOOT SIN CERTIFICADOS

Ángel Francisco Zato del Corral

JUNIO 2016

COMUNICACIONES SEGURAS EN TIEMPO DE PRE-BOOT SIN CERTIFICADOS

AUTOR: Ángel Francisco Zato del Corral
TUTOR: Javier Téllez Chacón
PONENTE: Jorge E. López de Vergara Méndez

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2016

RESUMEN

En este Proyecto se aborda la búsqueda de una solución adecuada para securizar las comunicaciones entre equipos informáticos en las que los extremos participantes de la comunicación cuentan con una serie de limitaciones importantes entre las que cabe destacar que no cuentan con un sistema operativo que facilite las funciones de red y que tampoco han de utilizar certificados digitales para garantizar la autenticidad de los extremos.

La realización de este proyecto abarca el análisis, diseño y construcción de un prototipo que servirá de base para la posterior productización de un sistema completo.

El problema a resolver, y a su vez el presente proyecto, consta esencialmente de dos partes bien diferenciadas. En primer lugar se busca una solución para dotar a los extremos de la comunicación de un soporte de funciones red completo cuando no se cuenta con un sistema operativo o este no ha sido cargado aún. Posteriormente, se pone el foco en asegurar la comunicación utilizando Criptografía Basada en Identidad, una aproximación criptográfica ciertamente minoritaria pero que ha experimentado un auge notable en los últimos años. Esta solución va a permitir el cifrado de la información trasmitida y la autenticación mutua de los extremos intervinientes sin recurrir al uso de certificados digitales.

Palabras clave:

Criptografía Basada en Identidad, sistema operativo, PXE, pre-boot, certificados digitales, comunicaciones, autenticación mutua, cifrado.

ABSTRACT

This project is about finding a suitable solution to secure communications between computer systems where the participants of the communication have strong limitations as neither having an operating system which provides network functions support nor being allowed to use digital certificates for guaranteeing end's authenticity.

The project goes through the analysis, design and implementation of a prototype which will be later used for building the complete solution.

The problem resolved, and the project itself, consists of two main parts. First, we focus on finding a solution to provide the communication ends with full network capabilities when the operating system is missing or has not been loaded yet. Then, we move forward to the task of making the communication secure based on Identity Based Cryptography, a minority cryptographic approach which has considerably evolved over the last few years. This kind of solution allows both the encryption of the information exchanged and the authentication of the communication's ends not requiring the use of digital certificates.

Key words:

Identity Based Cryptography, operating system, PXE, pre-boot, digital certificates, communications, mutual authentication, encryption.

AGRADECIMIENTOS

Ha sido grande, ímprobo y, finalmente, gratificante el esfuerzo realizado para llegar hasta aquí. Innumerables obstáculos, retorcidos puzles de difícil solución y una complejidad tecnológica mucho mayor de la prevista inicialmente constituyen el muro que he tenido que afrontar para completar este proyecto de fin de carrera y, con él, completar los estudios de Ingeniero de Telecomunicaciones.

Dice nuestro refranero que *es de bien nacidos ser agradecidos*, y a eso precisamente dedicaré las siguientes líneas.

Gracias a Javi, mi tutor, por su constante ayuda y sabias indicaciones a la hora de tomar delicadas decisiones. También a Jorge, por ofrecerse a ejercer el rol de ponente sin tan siquiera conocerme cuando acudí a él en busca de ayuda.

Gracias a mis amigos y compañeros de trabajo José y Julián por haberme echado una mano cuando las cosas se tornaban demasiado negras. También a Igor, que siempre está ahí para echar una mano cuando se le necesita. ¡Os debo una cena como es debido! Pero una de verdad, no una en la Churrasquita.

Gracias a mis jefes Pedro y Juanje por las facilidades que me han dado para realizar el proyecto y también por sus sabios consejos e indicaciones.

Gracias a mis padres y hermanos, en especial a los primeros, quienes no han dejado de animarme y empujarme durante toda la carrera para que llegara hasta el final y no colgara los guantes por el camino. Mención especial para mi hermano pequeño, Zoltan, el lingüista de la familia, por haberme ayudado con la corrección ortográfica y gramatical de esta memoria para que luzca más y mejor.

Por último, mi agradecimiento eterno a los pilares fundamentales de mi vida: mi mujer, Sandra, y mi hijo, Ángel. Ellos son la esencia de mi vida y sin lugar a dudas han sido los principales damnificados por la realización de este proyecto. Todo el tiempo que le he tenido que dedicar les pertenecía a ellos. Os lo compensaré.

CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

GLOSARIO

ATM (Automated Teller Machine)

Cajero automático.

Broadcast

Petición o mensaje que se envía a todos los elementos de una red de comunicaciones.

Bootable

Arrancable, aunque no suele traducirse al español. Indica que una imagen software sirve para arrancar un sistema operativo.

Boot Manager/Boot Helper

Pequeño programa encargado de arrancar un sistema operativo.

BOOTP (Bootstrap Protocol)

Protocolo utilizado para obtener automáticamente una dirección IP normalmente antes de la carga del sistema operativo.

Certificado digital

Fichero generado por una entidad de certificación que vincula ciertos datos de identidad a una persona o equipo informático.

Chipset

Conjunto de componentes electrónicos que controlan el flujo de datos entre un procesadores, memorias y otros dispositivos periféricos.

Criptoanalista

Persona especializada en el arte de descifrar mensajes cifrados.

Criptografía

Arte de diseñar algoritmos de cifrado.

DHCP (Dynamic Host Configuration Protocol)

Protocolo que permite a un equipo obtener una dirección IP dinámica.

FIPS (Federal Information Processing Standard)

Estándar federal de procesamiento de la información del Gobierno de los Estados Unidos de América.

Gusano

Programa malicioso que tiene la propiedad de duplicarse a sí mismo y propagarse entre sistemas informáticos sin necesidad de que intervenga una persona.

Hipervisor

Sistema que, aplicando diversas técnicas de virtualización de hardware, permite la ejecución simultánea de varios sistemas operativos sobre un único ordenador.

HSM (Hardware Security Module)

Módulo de seguridad basado en hardware que proporciona almacenamiento seguro para claves criptográficas al igual que procesamiento de algoritmos criptográficos.

IDS/IPS (Intrusion prevention/detection system)

Sistemas informáticos de prevención/detección de intrusiones.

JNA (Java Native Access)

Librería que proporciona a los programas desarrollados en Java un acceso fácil y sencillo a librerías de código nativo.

Kernel

Núcleo monolítico que constituye la parte esencial de un sistema operativo o programa complejo.

Kiosco

Equipo informático situado en un lugar público que ofrece una serie de funcionalidades a los usuarios.

Leak

Término que se emplea en programación para denotar la pérdida o el consumo inadecuado de un cierto recurso (típicamente memoria) debido a un error de codificación o a una gestión indebida de dicho recurso.

Máquina Virtual de Java

Conocida habitualmente por su singlas en inglés JVM, es el entorno de ejecución sobre el que se ejecutan los programas escritos en lenguaje Java.

MIT (Massachusetts Institute of Technology)

Instituto de Tecnología de Massachusetts.

Multivendor

Término empleado para referirse de forma genérica al software o hardware que puede provenir de distintos fabricantes. La traducción al castellano sería *multifabricante*, pero no suele utilizarse, por lo que se prefiere el término en inglés.

NBP (Network Bootstrap Program)

Programa de arranque en red. Se trata de un pequeño programa que amplía las funciones del cliente básico del PXE dotando al equipo de un soporte de comunicaciones más completo.

NIST (National Institute of Standards and Technology)

Instituto Nacional de Estándares y Tecnología de los Estados Unidos. Es considerado una referencia muy importante en materia de estandarización, innovación y regulación en múltiples disciplinas tecnológico-industriales.

NSA (National Security Agency)

Agencia Nacional de Seguridad de los Estados Unidos de América

OTP (One Time Password)

Clave que es válida para un único uso.

PC (Personal Computer)

Computadora personal. En castellano se prefiere el término *ordenador personal* o simplemente *ordenador*.

PFC

Proyecto de final de carrera.

PIN (Personal Identification Number)

Clave numérica que se utiliza para autenticar un usuario ante un determinado sistema.

PKG (Private Key Generator)

Centro Generador de las Claves privadas de los usuarios de un sistema criptográfico basado en identidad. Citado también en esta memoria como Centro Generador de Claves o simplemente Centro Generador.

POS (Point Of Sale)

Punto de Venta. Equipo informático que sustituye las antiguas cajas registradoras en los comercios y que es capaz de aceptar tarjetas de crédito para realizar transacciones bancarias.

PXE (Preboot eXecution Environment)

Entorno de arranque previo a la carga del sistema operativo.

RAM (Random Access Memory)

Memoria volátil de acceso aleatorio. En ella se coloca el código de los programas a ser ejecutados por el procesador.

Rootkit

Software, típicamente malicioso, que se instala a muy bajo nivel en un equipo informático y que altera el funcionamiento normal del equipo pasando desapercibido para la totalidad de las aplicaciones e incluso el sistema operativo del equipo.

Seguridad lógica

Aplicación de barreras y defensas para proteger datos, programas y recursos informáticos sin entidad física frente a accesos indebidos o no autorizados.

SSL/TLS (Secure Socket Layer/Transport Layer Security)

Capa de seguridad que se añade a las comunicaciones TCP/IP para hacerlas más seguras proporcionando autenticación y cifrado gracias al uso de criptografía asimétrica y certificados digitales.

TCP/IP (Transmission Control Protocol/Internet Protocol)

Protocolo de Control de la Transmisión/Protocolo de Internet. Conjunto de protocolos que funciona como base para otros servicios o protocolos de más alto nivel que requieren comunicación entre equipos interconectados.

TFTP (Trivial File Transfer Protocol)

Protocolo básico de transferencia de archivos basado en UDP.

TPM (Trusted Platform Module)

Dispositivo hardware especializado que se incluye en los PC modernos para el almacenamiento y el procesado de claves criptográficas.

TPV (Terminal Punto de Venta)

Equipo informático situado en un establecimiento comercial que ayuda en la gestión del negocio.

Troyano

Programa malicioso que se presenta ante el usuario como un problema legítimo o inofensivo pero que al ser ejecutado habilita un acceso remoto al equipo infectado.

UDP (User Datagram Protocol)

Protocolo del nivel de transporte no orientado a conexión.

Virus

Programa malicioso que se ejecuta en un equipo informático con la intención de producir un cierto daño, alterar el funcionamiento normal o manipular determinados datos o aplicaciones.

Windows

Familia de sistemas operativos con entorno gráfico del fabricante Microsoft.

1. INTRODUCCIÓN

Continuamente llegan hasta nuestros oídos noticias de ataques y fraudes perpetrados contra dispositivos electrónicos de los que todos somos usuarios habituales, tales como ordenadores, tablets, teléfonos móviles, etc. Virus, troyanos, gusanos o ataques de denegación de servicio son solo algunos de los ataques más comunes y, en mayor o menos medida, el usuario medio está familiarizado con estos términos. Estos tipos de ciberataques quedan englobados dentro de un conjunto más amplio que podríamos denominar ataques contra la seguridad lógica de los sistemas, puesto que el objetivo del ataque no es el hardware del dispositivo, sino el software o los datos que almacenan y gestionan, es decir, aquello que es intangible y no tiene entidad física.

Sin embargo, en la actualidad un alto porcentaje de la población somos también usuarios de ciertas máquinas especiales que forman parte de nuestra vida cotidiana, sin ser conscientes de que dichos equipos, y por consiguiente nuestros datos, están también expuestos a ataques de seguridad lógica. Estos equipos son, entre otros, los cajeros automáticos o ATM, los kioscos, los POS, TPV, etc.

Desde el punto de vista funcional, no son más que máquinas capaces de ofrecer una serie de servicios a sus usuarios. Retirada de efectivo, consulta de saldo, impresión y retirada de entradas compradas por Internet, adquisición de billetes de Metro o Cercanías, compra de gasolina en surtidores de autoservicio, navegación por Internet, etc., son solo algunos ejemplos de la amplia gama de opciones que ofrecen a sus usuarios.

Si nos centramos en el hardware de estos equipos, encontramos que se trata de PC convencionales, habitualmente de bajas prestaciones, a los que se han conectado una serie de dispositivos periféricos especializados, tales como lectores de tarjetas de crédito/débito, impresoras de billetes o recibos, cajas registradoras, datafonos, cajas fuertes, dispensadores de efectivo, etc.

Por otra parte, estos equipos se encuentran interconectados entre sí, con un host autorizador de transacciones o con los servidores asociados a las aplicaciones y servicios correspondientes a través de una red de comunicaciones. A día de hoy, las redes que interconectan este tipo de máquinas entre sí y con el resto de elementos son en su mayoría redes TCP/IP. Estos equipos terminales que forman parte de una red de comunicaciones reciben típicamente el nombre genérico de *endpoints*. En adelante, se empleará el término *endpoint* para referirse al conjunto de estos equipos (ATM, kioscos, TPV, etc.).

Aunque no es aplicable al cien por cien de los casos, estos entornos de endpoints presentan habitualmente una serie de características comunes que los hacen diferentes de lo que podría ser una red de PC de oficina. Estas características se resumen brevemente en las siguientes:

- Son equipos de bajas o medias prestaciones equipados habitualmente con sistemas operativos Windows (NT, XP o 7).
- Las redes TCP/IP que los interconectan entre sí pueden ser públicas, privadas o mixtas. En muchas ocasiones el ancho de banda es muy limitado o se paga por uso de la red.
- El software instalado es estático. No se actualiza continuamente pudiendo permanecer en el mismo estado durante meses o inclusos años. Se busca potenciar al máximo la disponibilidad del endpoint

(tiempo que está operativo para el usuario) y esto se traduce necesariamente en hacer el mínimo número de cambios en el software instalado.

- Los dispositivos especializados que se conectan al PC elevan considerablemente el precio del conjunto. Esto hace que las empresas o entidades propietarias de estos equipos los expriman hasta el último día de su vida útil antes de remplazarlos por equipos modernos. Es bastante habitual encontrar endpoints dando servicio en la calle con más de veinte años desde su puesta en explotación.

1.1. AMENZAS Y SOLUCIONES DE SEGURIDAD PARA REDES DE ENDPOINTS

Si bien las clásicas amenazas de virus, troyanos, gusanos, etc., atentan potencialmente contra las redes de endpoints, con mayor frecuencia aparecen ataques lógicos más sofisticados y singulares que provienen del propio personal de las empresas que los instalan y/o mantienen, o de personas con conocimiento profundo del hardware y el software contra el que realizan el ataque.

Las particularidades expuestas sobre el hardware, el software y la red de comunicaciones, así como el tipo de ataques lógicos que sufren los *endpoints*, hacen que las soluciones tradicionales de seguridad como antivirus, anti-malware, IDS/IPS, etc., sean ineficaces. Esto da lugar a aplicaciones de seguridad más especializadas, basadas principalmente en tecnología de listas blancas que pueden proporcionar un control férreo del entorno de ejecución y comunicaciones. Por otra parte, como contramedida ante los últimos vectores de ataque acontecidos recientemente, también se hace necesario proteger los sistemas de ficheros locales con soluciones de cifrado integral de discos duros.

1.2. MOTIVACIÓN

1.2.1. COMUNICACIONES EN PRE-BOOT. ¿QUÉ SON Y PARA QUÉ SE USAN?

Mientras que la utilidad de las comunicaciones cuando el endpoint está operativo y funcionando en modo ordinario es incuestionable, existe otro tipo de comunicaciones que también aparecen con cierta frecuencia (relativamente baja) en el mundo del endpoint. Se trata de las denominadas comunicaciones en pre-boot, esto es, las comunicaciones que se establecen cuando el sistema operativo aún no ha sido cargado y la secuencia de inicio del equipo acaba de ser iniciada. Cuando un PC es alimentado, comienza la secuencia de arranque. El primer código en ser ejecutado por el procesador está almacenado en la memoria persistente de la BIOS del equipo. Este código típicamente está preparado para buscar un dispositivo desde el que arrancar un sistema operativo. Este dispositivo es habitualmente el disco duro local del PC, pero no tiene por qué ser así. El sistema operativo podría encontrarse en una unidad de CD/DVD-ROM, un disco extraíble USB o simplemente en un equipo remoto accesible a través de la red de comunicaciones.

A continuación se enumeran algunas utilidades de las conexiones en pre-boot típicamente empleadas en las redes de endpoints:

- Inicio de un sistema operativo diferente al instalado en el disco duro local. Permite por ejemplo cargar sistemas de diagnóstico.

- Reinstalación del software del endpoint. Ante determinadas circunstancias, el software de pre-boot puede recibir la instrucción de reinstalar el sistema operativo local, por ejemplo quemando una nueva imagen proveniente de la red o de algún otro dispositivo local (disco USB, CD/DVD-ROM, segundo disco duro local, otra partición en el disco principal, etc.).

- Recibir algún parámetro especial que modifique la carga o puesta en servicio del endpoint, por ejemplo dando entrada al modo supervisor.

- Recibir algún dato o parámetro que reconfigure la actual tabla de particiones del disco y, por lo tanto, el modo de arranque.

- Recibir una clave que permita el acceso a algún contenido protegido del endpoint o de la red.

Una de las causas para el escaso despliegue de soluciones pre-boot es la ausencia de soporte multivendor, es decir, la ausencia de soluciones que puedan funcionar sobre hardware de distintos fabricantes. Esto es debido principalmente a que las implementaciones de PXE se encuentran típicamente en el firmware de la BIOS o en el firmware de las propias tarjetas de red. Tanto en un caso como en el otro, el soporte encontrado es muy particular. Algunas BIOS no incorporan soporte para PXE y otras muchas lo hacen de manera parcial o con defectos de funcionamiento que solo pueden ser solventados mediante la actualización del firmware de la BIOS, tarea que en muchas ocasiones es impracticable. Con las implementaciones presentes en las propias tarjetas de red ocurre prácticamente lo mismo.

1.3. OBJETIVOS

1.3.1. COMUNICACIONES SEGURAS EN PRE-BOOT SIN CERTIFICADOS

Las empresas y entidades que hacen uso de los sistemas de pre-boot son conscientes de que estos introducen ciertos problemas de seguridad. ¿Qué sucedería si el endpoint, en lugar de conectarse a un servidor fiable para descargar un sistema operativo legítimo, estuviera conectándose a un servidor malintencionado? El endpoint podría recibir y ejecutar código malicioso cuya ejecución podría traducirse en un daño irreparable para el equipo, en la pérdida de información sensible o simplemente en un daño en la imagen de la compañía propietaria del sistema. Por otro lado, ¿qué sucedería si un tercer equipo no autorizado, haciéndose pasar un endpoint de la red, se presentara ante el servidor y descargara una imagen, una clave o cierta información sensible? En este caso, ese equipo estaría apropiándose de forma indebida de ciertos datos valiosos que no le corresponden.

La solución a estos problemas pasa por asegurar las comunicaciones que tienen lugar en tiempo de pre-boot garantizando los siguientes dos requisitos fundamentales:

- **Autenticación de los extremos:** es necesario que ambos extremos puedan reconocerse como extremos legítimos de la comunicación.

- **Cifrado:** el intercambio de información entre los extremos no debe realizarse en claro, sino de forma confidencial, de modo que únicamente los extremos de la comunicación tengan la capacidad de entender los datos intercambiados.

Existen algunas soluciones de PXE que soportan comunicaciones en pre-boot con SSL/TLS que posibilitan la autenticación de uno de los extremos y el cifrado de la comunicación. Sin embargo no posibilitan la autenticación mutua y además requieren de la instalación de certificados. De este modo, instalando el certificado, típicamente en el extremo Servidor, los endpoints podrían autenticarlo antes de descargar ninguna información de él. Sin embargo, el Servidor no tendría la posibilidad de autenticar a los endpoints, ya que estos no contarían con un certificado.

La instalación de un certificado es un endpoint no es una solución aceptable en algunos entornos por los siguientes motivos:

- Generar un certificado distinto para cada uno de los endpoints (a veces varios miles en la misma red) supone una carga de trabajo adicional para el personal de sistemas/infraestructuras.
- Los certificados deben ser emitidos con caducidad no superior a uno o dos años, como máximo. La actualización de estos supone aún más tiempo de gestión y mantenimiento que las empresas tienen que dedicar.
- Existen problemas generales asociados al uso de certificados, como el uso de listas de revocación no actualizadas, que hacen que estos no tengan buen cartel en ciertos entornos de *endpoint*.

Analizando todas las variables y consideraciones expuestas, se necesita un sistema adecuado de comunicaciones seguras en pre-boot. La solución debe reunir los siguientes requisitos:

- Ha de ser **multivendor/multicore:** debe poder instalarse en *endpoints* de distintos fabricantes y modelos, independientemente de su año de fabricación y versión. Por su parte, el término *multicore,* en este contexto, no hace referencia ni al número ni al tipo de núcleos del procesador, sino a todos los componentes hardware del PC, esto es, el conjunto formado por la BIOS, la placa base, la controladora de discos, la tarjeta gráfica, la tarjeta de red, el procesador, el disco duro, etc.

- Debe tener **soporte para UDP:** dadas las limitaciones de ancho de banda en algunos enlaces, en algunas ocasiones se prefieren las conexiones UDP a las TCP. En realidad, no es habitual hacer uso de UDP puro, sino de extensiones ligeras de este protocolo que permiten, por ejemplo, garantizar la entrega de los datagramas o cifrar su contenido sin que suponga una penalización excesiva en ancho de banda.

- Debe proporcionar un sistema de **autenticación de los extremos sin requerir la instalación de certificados digitales.** Para lograr tal objetivo se va a utilizar la denominada *Criptografía Basada en Identidad*, que es un tipo de criptografía de clave pública en la que se utiliza como clave pública la representación en cadena de caracteres de un objeto o persona. La clave privada asociada a dicha clave pública es generada por un *Generador de Claves Privadas*, que ha de ser fiable.

- Debe **cifrar las comunicaciones** para garantizar la confidencialidad de los datos intercambiados.

El objetivo final de este PFC es la realización de un prototipo que sirva como sólido punto de partida para el desarrollo del sistema final que pueda ser incluido en los productos de seguridad de cualquier endpoint.

1.4. ESTRUCTURA DE LA MEMORIA

El contenido de la presente memoria está estructurado en una serie de capítulos que podríamos agrupar en dos grandes bloques. En el primer bloque, formado por los capítulos 2 y 3, abordamos la investigación, estudio y establecimiento de las bases para todo el desarrollo de lo que va a venir a continuación. En el segundo bloque, compuesto por los capítulos 4, 5 y 6, analizamos las fases de diseño de la solución, desarrollo y validación. En los capítulos siguientes analizamos las conclusiones, incluimos una serie de apéndices para completar o destacar algunos aspectos analizados en apartados anteriores y realizamos un análisis de los costes asociados al proyecto.

Veamos con un poco más de detalle lo qué vamos a encontrar en cada uno de los capítulos.

El capítulo 1 de la memoria, del que este apartado forma parte, constituye la introducción al presente proyecto. En él se presentan los objetivos y la motivación que justifican y dan sentido a la realización del proyecto. Asimismo, se establecen los requisitos que la solución a diseñar debe cumplir.

En el capítulo 2 comenzaremos por ilustrar qué son las comunicaciones en pre-boot y para qué se usan. Presentaremos el estándar PXE como piedra angular que rige este tipo de comunicaciones. A continuación, centraremos nuestro estudio en la elección de una solución adecuada que proporcione a los endpoints objeto de estudio el mejor soporte para comunicaciones en pre-boot, siempre sin perder de vista que pretendemos modificarla para añadirle el soporte criptográfico necesario. Seguidamente veremos cómo instalar dicho soporte de comunicaciones en los endpoints sin afectar al sistema operativo que ya estuviera presente en el endpoint.

En el capítulo 3 se realiza una breve introducción a la criptografía moderna, el álgebra modular y la tecnología de curvas elípticas para dotar al lector poco ducho en la materia de la base necesaria para poder seguir el proyecto. Seguidamente presentaremos los conceptos de Emparejamiento Bilineal y Criptografía Basada en Identidad. Con las bases asentadas, en el capítulo 4 diseñamos el prototipo de sistema criptográfico adecuado para securizar las comunicaciones en pre-boot.

El capítulo 5 incluye la implementación del sistema criptográfico diseñado en el capítulo anterior y que da respuesta a las necesidades y requerimientos planteados por el problema inicial. En este capítulo nos detendremos brevemente a revisar algunos aspectos del desarrollo que hemos considerado relevantes por su originalidad o por no estar presentes habitualmente en otros proyectos de desarrollo de software. También revisaremos en este capítulo los medios utilizados para la implementación de la solución.

En el capítulo 6 se enumeran y describen las pruebas realizadas para validar la solución construida.

El capítulo 7 presenta un resumen del proyecto, una serie de conclusiones finales y el trabajo futuro a realizar.

El capítulo 8 recoge las referencias y bibliografía consultadas para la realización del proyecto.

En los capítulos 9, 10 y 11 se incluyen una serie de apéndices que están dedicados a revisar y reforzar algunas ideas o conceptos y a profundizar o ampliar detalles de alguno de los aspectos o temas expuestos en el cuerpo de la memoria.

Finalmente, en los capítulos 12 y 13 se incluyen, respectivamente, el pliego de condiciones y el presupuesto del proyecto.

Todos los capítulos desde el segundo al séptimo finalizan con un pequeño apartado de resumen y conclusiones en el que se repasan las ideas principales expuestas en el apartado y se ilustra, cuando procede, su relación o nexo con otros capítulos de la memoria.

2. COMUNICACIONES EN PRE-BOOT

Cómo ya se ha esbozado en la sección 1.2, entendemos por comunicación en tiempo de pre-boot, o simplemente comunicación en pre-boot, aquella que tiene lugar antes de que se produzca la carga del sistema operativo en el endpoint. No perdamos que vista que un endpoint no es más que un PC convencional con una serie de dispositivos periféricos especiales conectados.

Para entender mejor las implicaciones y sobre todo las limitaciones que tienen este tipo de comunicaciones, comenzaremos por describir cómo sucede la secuencia de arranque de un PC.

2.1. LA SECUENCIA DE ARRANQUE

En el momento en que se pulsa el interruptor de encendido del PC, la fuente de alimentación proporciona corriente a la placa base, el microprocesador, las unidades ópticas, los discos duros, ventiladores, etc. El microprocesador toma el control y ejecuta el código de la BIOS (Basic Input Output System) que ha sido cargado en la posición FFFFh de la memoria RAM del equipo. La BIOS es por tanto el primer programa que se ejecuta, y se encuentra almacenado típicamente en una memoria persistente de solo lectura instalada dentro del PC. Los PC modernos cuentan con BIOS implementadas en memoria flash, lo cual, por una parte, hace posible su actualización, pero, por otra, presenta algunos inconvenientes, como la posibilidad de ser infectadas con virus o rootkits específicos.

Una vez en ejecución, el código de la BIOS comienza a realizar comprobaciones y chequeos de los dispositivos presentes. Además, una de las primeras cosas que el código de la BIOS hace es localizar la tarjeta gráfica y arrancar el sistema de video para poder proporcionar una salida legible en el caso de que haya una pantalla conectada al PC. Después, prosigue con los chequeos e identificación de los dispositivos. Cuando el código de la BIOS ha finalizado con todas estas tareas, llega el momento de localizar un dispositivo de almacenamiento al que pasar el testigo para que la secuencia de arranque continúe. Lógicamente, no vale con localizar un dispositivo de almacenamiento cualquiera, sino uno que contenga un código de arranque, también conocido como código *bootstrap* o código MBR. Además, la BIOS debe respetar el orden de búsqueda establecido para seleccionar el dispositivo de almacenamiento correcto de entre todos aquellos que son susceptibles de contener código MBR. Típicamente, la BIOS puede ser configurada para localizar y arrancar, en el orden preestablecido, el código de arranque contenido en discos duros, memorias o discos USB, unidades ópticas (CDROM o DVDROM) o tarjetas de red. Abordaremos más adelante qué significa el arranque por red, y por el momento nos centraremos en analizar los otros casos, que en esencia son el mismo.

El dispositivo de almacenamiento seleccionado por la BIOS para continuar con la secuencia de arranque será aquel que, atendiendo al orden de prioridad ya comentado, contenga la bandera de código de *bootstrap* en el sector de arranque de la unidad. Esta bandera no es más que la pareja de bytes **55AA**h. El sector de arranque de una unidad de almacenamiento es siempre el primer sector de esta, de tamaño 512 de bytes. A este sector también se le conoce como MBR (Master Boot Record) por motivos obvios. Toda aquella unidad que no contenga un MBR válido será descartada y la BIOS pasará a analizar la siguiente.

Una vez encontrada la unidad adecuada, el código del sector de arranque se colocará en la memoria RAM del equipo y el procesador comenzará a ejecutarlo. Obviamente este código, de tamaño inferior a 512 bytes, no puede albergar un programa complejo, y mucho menos un sistema operativo. Lo que contiene este código es una pequeña porción o etapa del programa que se denomina *boot helper* o *boot manager* y que tiene la función de localizar dentro de la unidad la parte restante de este programa *boot helper.* Es esta segunda etapa, de mucho mayor tamaño, la que implementa el verdadero cargador del sistema operativo. A modo de ejemplo, se expone el código MBR que instala el sistema operativo Windows XP en el primer sector de un disco duro convencional.

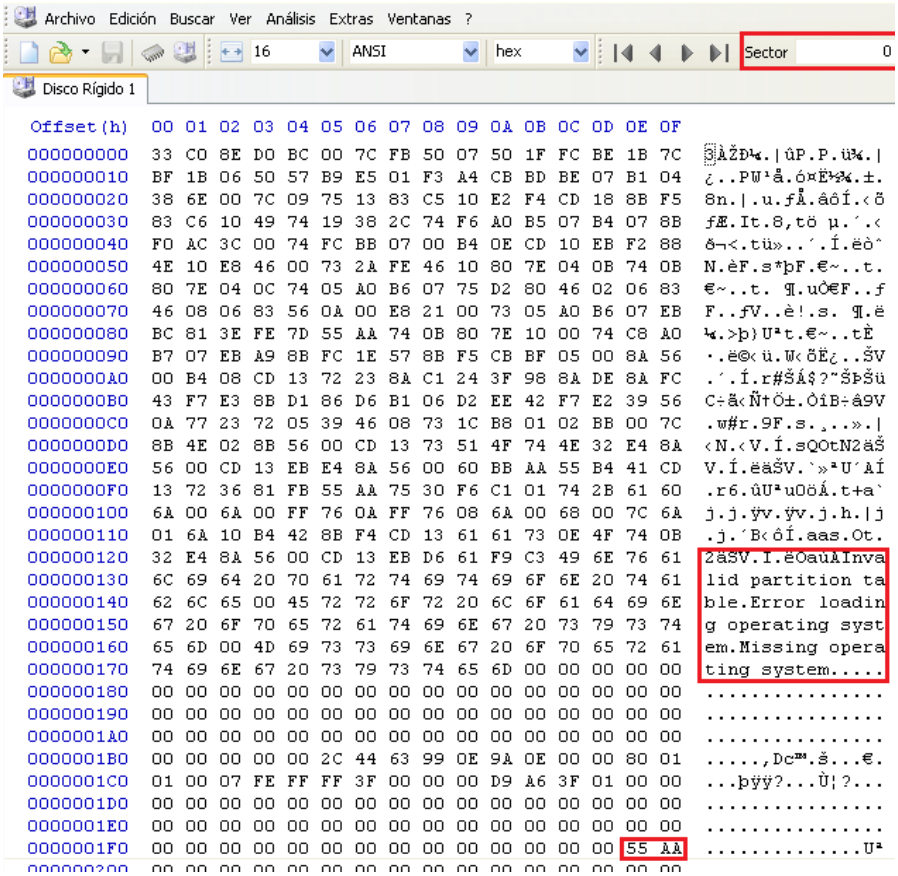


Ilustración 1 - Código MBR original de Windows XP

Como se puede observar en la imagen, el código MBR ocupa poco más de 400 bytes y se puede identificar claramente la cadena “Invalid partition table. Error loading operating system. Missing operating system”. Precisamente ese mensaje se mostrará por pantalla si este pequeño código no es capaz de localizar el fichero NTLDR en ninguna de las particiones primarias del disco. El fichero NTLDR es el que contiene el *boot helper* completo, y será finalmente el encargado de lanzar la carga del sistema operativo Windows XP. Para el curioso lector, el *boot helper* de Windows 7 está contenido en el fichero BOOTMGR y habitualmente se reserva una partición primaria de pequeño tamaño para albergarlo.

Nos hemos detenido a analizar en detalle lo que es el código MBR y dónde se instala porque más adelante veremos que va a jugar un papel fundamental a la hora de conseguir un soporte de red completo en pre-boot.

2.2. PXE

Dejemos a un lado la secuencia de arranque de nuestros endpoints –la retomaremos pronto– y centrémonos de nuevo en las comunicaciones. Cuando se habla de comunicaciones en pre-boot, el término PXE aparece inmediatamente en escena. PXE son las siglas de Preboot eXecution Environment, lo que traducido a la lengua de Cervantes viene a significar "entorno de ejecución pre-arranque (del sistema operativo)".

PXE es un estándar que permite a equipos conectados a través de una red de comunicaciones interactuar entre ellos o ser administrados remotamente antes de que se produzca la carga del sistema operativo. Originalmente fue diseñado para que equipos equipados con una tarjeta de red pudieran descargar y ejecutar imágenes software desde servidores, accediendo a estos a mediante el uso de protocolos cliente-servidor estandarizados tales como BOOTP, TFTP o DHCP.

La primera especificación de PXE fue descrita por Intel inicialmente como parte de una especificación más amplia, denominada Wired For Management 1.0, en 1997. La versión 2.0 de PXE fue también publicada como parte de la misma especificación actualizada, Wired For Management 2.0, a finales de 1998. Finalmente, la versión 2.1 de PXE vio la luz en septiembre de 1999 y se convirtió en un estándar adoptado por todos los fabricantes de tarjetas de red. De hecho, es prácticamente imposible encontrar una tarjeta de red fabricada a partir del año 1999 que no incluya firmware con soporte de PXE.

2.2.1. FUNCIONAMIENTO

Sin entrar en excesivo detalle ilustraremos a continuación cómo funciona PXE. Recordemos que la idea que hay detrás de este estándar es la de que sistemas equipados con una tarjeta de red puedan descargar imágenes ejecutables (que contienen sistemas operativos o programas) desde ciertos servidores de imágenes software. Veamos cómo funciona esto paso por paso.

En primer lugar, el cliente PXE que se ejecuta en el endpoint necesita obtener una dirección IP, una máscara de red y una puerta de enlace con las que poder interactuar con el servidor de imágenes o cualquier otro equipo de la red. Para que esto suceda es necesario que exista un servidor DHCP en la red. Por tanto, el cliente de PXE comenzará por lanzar una petición broadcast de descubrimiento del servidor DHCP. El servidor DHCP, si es que hay uno presente en la red, contestará al cliente indicándole la dirección IP y el resto de parámetros de configuración de red TCP/IP que ha de utilizar en adelante.

Lógicamente, no es suficiente con que el cliente reciba estos datos, sino que necesita recibir también alguna indicación que le permita conocer qué imagen de software debe descargar y desde qué servidor debe hacerlo. Para ello el estándar PXE introduce el concepto de servidor proxy-DHCP. Este servidor será el encargado de proporcionar al cliente PXE la dirección IP del servidor TFTP y el nombre de la imagen que debe descargar y arrancar.

Este esquema de servidor DHCP convencional más servidor proxy-DHCP facilita la integración de servidores de imágenes software en aquellas redes donde ya existe un servidor DHCP.

Por otra parte, es habitual que el servidor PXE no proporcione directamente al cliente una imagen de un sistema operativo grande y complejo, sino que le proporciona una imagen que contiene un pequeño programa que recibe el nombre de NBP. Este NBP cuenta con funciones ampliadas respecto al cliente

básico de PXE. Entre estas funciones cabe destacar el soporte de protocolos de comunicación más evolucionados como HTTP o NFS que van a permitir trasferencias más fiables con los servidores que finalmente proporcionan el sistema operativo final. TFTP, al estar basado en el protocolo de transporte UDP, no se considera un protocolo fiable.

El cliente básico de PXE se encuentra habitualmente en el firmware de las tarjetas de red y se espera que la implementación sea idéntica independientemente del fabricante y modelo de la tarjeta de red. Este cliente se caracteriza por tener una huella muy baja lo que facilita su integración en el firmware de las tarjetas. El cliente PXE contiene en esencia una implementación de los protocolos UDP, TFTP y DHCP.

El siguiente esquema ilustra el funcionamiento de un cliente PXE descargando una imagen "bootable" de un sistema operativo.

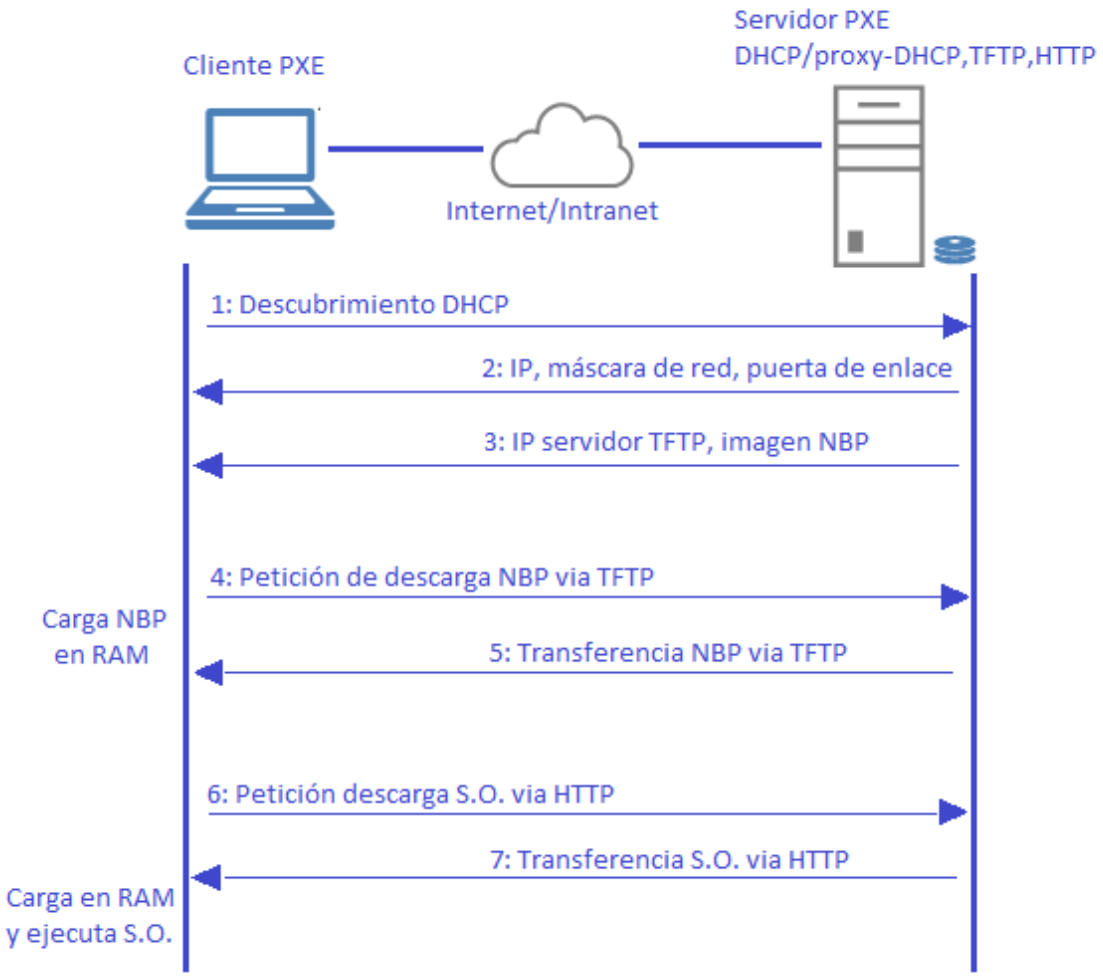


Ilustración 2- Esquema funcionamiento PXE

2.2.2. SOPORTE DE PXE PARA LOS ENDPOINTS

Una vez estudiados los conceptos básicos que hay detrás de una comunicación en pre-boot es el momento de encontrar el mejor soporte de PXE para los endpoints que son objeto de estudio en este proyecto.

Acabamos de ver que la mayor parte de las tarjetas de red incluyen de forma nativa un cliente de PXE desde finales de los años 90. Sin embargo no será esta la opción por la que nos decantaremos para proporcionar soporte de comunicaciones en pre-boot por los siguientes motivos:

- No es fácil su actualización. No todas las tarjetas de red permiten actualizar su firmware y para aquellas que sí lo hacen –las basadas en memoria flash-, es muy complicado hacerlo de manera remota dado que habría que enviar previamente el firmware actualizador. Además este tipo de programas de actualización pueden dejar el equipo completamente colgado en caso de producirse algún problema. De producirse esto habría que realizar una intervención física para poder recuperar el endpoint. Todo esto atenta claramente contra la disponibilidad del endpoint, que debe ser la máxima posible.

- Existen multitud de entornos donde los endpoints pueden tener muchos años y sus tarjetas de red haber sido fabricadas con anterioridad a 1999. En esos casos no contarían con soporte para PXE. Uno de los requisitos indispensables en este proyecto es que la solución alcanzada sea aplicable a todos los endpoints independientemente de su fabricante, modelo, arquitectura o año de fabricación de sus componentes.

Una vez descartado el soporte nativo en firmware que se encuentra en el firmware de las tarjetas de red, la pregunta es: ¿Qué alternativas hay? ¿Cuál es el estado del arte?

Existen algunos proyectos que implementan soporte de PXE como cargadores de arranque. Para eludir la confusión que puede causar referirnos a estas implementaciones como cargadores de arranque nos referiremos a ellos como *kernels de PXE*.

Un kernel de PXE es por tanto un fichero o conjunto de ficheros ejecutables que cargados en memoria proporcionan al equipo las funciones de un cliente PXE. Con respecto a la implementación en el firmware de las tarjetas de red estos kernels proporcionan las siguientes ventajas:

- Son fácilmente actualizables. Al tratarse de ficheros convencionales, actualizarlos es tan sencillo como remplazar el fichero/ficheros por uno/unos nuevos. Como además en los entornos de endpoints suele darse la presencia de herramientas de distribución de software, la actualización de estos kernels es prácticamente trivial.

- La mayoría de estos kernels son proyectos de código abierto por lo que se pueden adaptar y parametrizar a medida simplemente teniendo conocimientos del lenguaje de programación en el que han sido implementados.

- De serie incorporan soporte para múltiples protocolos de comunicación avanzados (HTTP, TCP, NFS, etc.) y no solamente los protocolos básicos asociados al cliente básico de PXE.

- Pueden instalarse en cualquier equipo que cuente con algún dispositivo de almacenamiento persistente. El tamaño de estos kernels suele ser muy pequeño (menos de una decena de megabytes).

- Existe una amplia comunidad de desarrolladores detrás de estos proyectos, lo que facilita su desarrollo y soporte. Continuamente incluyen nuevas características y drivers para dar soporte a nuevas tarjetas de red.

Pero como en todos los ámbitos de la vida, frente a esta lista de ventajas existe otra de inconvenientes que se expone a continuación:

- Es necesario compilar el kernel a partir del código fuente para obtener una imagen binaria ejecutable. Para usuarios poco experimentados o sin conocimiento de lenguajes de programación este hecho suele suponer un inconveniente. Existen binarios ya compilados para su uso general. En caso de no tener necesidad de modificarlos, estos binarios serán la solución más rápida.

- Para que puedan ser ejecutados es necesario subirlos a memoria en tiempo de pre-boot. Mientras que la ejecución de los clientes nativos en firmware de la tarjeta de red es inmediata (sólo es necesario habilitar una opción en la BIOS del equipo), la ejecución de estos kernels va a requerir de cargadores de arranque especiales.

- El soporte para tarjetas de red queda limitado a los drivers que hayan sido implementados como parte de estos proyectos. Afortunadamente, existen proyectos de kernels de PXE con un amplio soporte de tarjetas de red que además incorporan de forma continua nuevos drivers.

A pesar de estos inconvenientes, los requisitos especificados para este proyecto, nos llevan necesariamente a la utilización de estos kernels de PXE frente a los clientes nativos en firmware.

2.2.3. SELECCIÓN DE UN KERNEL DE PXE (ESTADO DEL ARTE)

Una vez tomada la decisión de optar por un kernel de PXE el siguiente paso será seleccionar uno adecuado para nuestro propósito. Existen diversas implementaciones de kernels de PXE pero nos interesan únicamente aquellas que sean de código abierto para poder incorporar más adelante el soporte criptográfico con el que se pretenden asegurar las comunicaciones en pre-boot. La securización de las comunicaciones será abordada más adelante, en lo que constituye el segundo de los dos grandes bloques de los que se compone este proyecto. Por el momento, nos centraremos en la selección del kernel de PXE que mejor se adapta a nuestras necesidades.

Existen varios proyectos de código libre que implementan clientes de PXE. Algunos de ellos de gozan de gran popularidad y prestigio dentro de la comunidad del software libre. Sin embargo, algunos de ellos no cumplen con el estándar PXE o lo solamente lo hacen de forma parcial. Para la elección del kernel adecuado hemos analizado los siguientes tres proyectos debido a su gran popularidad: iPXE, gPXE y PXELinux. Estos tres proyectos constituyen el estado del arte de las implementaciones de código abierto del estándar PXE.

Los dos primeros derivan de un modo u otro de un proyecto más antiguo llamado Etherboot. Tanto gPXE como iPXE son implementaciones completas del estándar PXE que pueden funcionar con independencia del firmware de la tarjeta de red aunque también pueden ser utilizados para construir ROMs de remplazo para sobrescribir la original de las tarjetas de red. Por su parte, PXELinux no es estrictamente un cliente de PXE atendiendo a la especificación, sino que se trata de un NBP, por lo que necesita de la existencia del firmware de PXE embebido en la ROM de la tarjeta de red para poder funcionar. Por ese motivo no es útil para nuestro propósito.

La mayor diferencia entre gPXE e iPXE la encontramos en el nivel de actividad de ambos proyectos. Mientras que gPXE fue descontinuado en 2010, iPXE se encuentra en la actualidad en constante desarrollo. De hecho, iPXE es considerado por muchos la continuación de gPXE ya que son los mismos desarrolladores lo que han implementado ambas soluciones.

La siguiente tabla ilustra una comparativa entre gPXE/iPXE, PXELinux y el firmware PXE presente en las tarjetas de red.

	iPXE/gPXE	PXELinux	Firmware PXE
Tipo	Cliente PXE avanzado	NBP	Cliente PXE básico
Instalación/actualización	Fácil	Fácil	Difícil
Soporte tarjetas	Amplio	Amplio	Individual
Precisa firmware PXE	No	Sí	-

Tabla 1 - Comparativa entre implementaciones de clientes de PXE

A continuación se describe cada una de las filas de la tabla para una mejor comprensión de la misma:

- **Tipo:** indica si se trata de un cliente PXE o de un Network Boot Program. Un NBP, como se ha comentado anteriormente, es un pequeño sistema operativo que el cliente básico de PXE descarga y que le permite contar con una serie de funciones avanzadas para la comunicación pre-boot de las que el cliente básico no dispone.

- **Instalación/actualización:** el valor de esta fila pretende reflejar cómo de fácil es poner en funcionamiento el software y cómo de costoso es realizar una actualización del mismo.

- **Soporte tarjetas:** indica si el software funciona exclusivamente con una tarjeta de red (para la que ha sido diseñado en exclusiva) o si por el contrario el mismo software da soporte a varias tarjetas.

- **Precisa firmware PXE:** indica si para poder funcionar el software necesita de la presencia de firmware PXE en la ROM de la tarjeta de red.

2.2.4. INSTALACIÓN DE iPXE EN EL ENDPOINT

La instalación del kernel de PXE elegido, iPXE, está compuesta de dos pasos. El primero de ellos es la obtención de la imagen ejecutable compilada a partir del código fuente. Veremos que se trata de un proceso sencillo y rápido. El segundo, consiste en la instalación del kernel compilado en el endpoint de tal

modo que pueda ser utilizado para iniciar comunicaciones en pre-boot. Se trata de un proceso más complicado que va a conllevar la instalación de software adicional.

2.2.4.1.COMPILACIÓN DE iPXE

El primer paso es la obtención del código fuente a ser compilado desde el sitio web del proyecto iPXE: http://ipxe.org/download

Existen imágenes binarias ya compiladas para que el usuario interesado pueda probar y experimentar de forma rápida un kernel de iPXE, sin embargo, y dado que pretendemos modificar el código para añadir seguridad a las comunicaciones, estaremos interesados solamente en la generación a partir del código fuente.

Para la realización de este proyecto se ha descargado la versión 1.0.0+ estable desde el sitio web del proyecto.

Una vez obtenidos los archivos fuente utilizaremos para su compilación un sistema Linux Ubuntu 12.0.4-LTS equipado con las siguientes aplicaciones y/o paquetes:

- gcc
- make
- perl
- syslinux
- Ficheros de cabeceras (.h) para: zlib, binutils and libiberty

Para compilar el kernel de iPXE simplemente nos situaremos en el directorio *src* y ejecutaremos el comando *make* desde un terminal tal y como se muestra en la siguiente imagen.

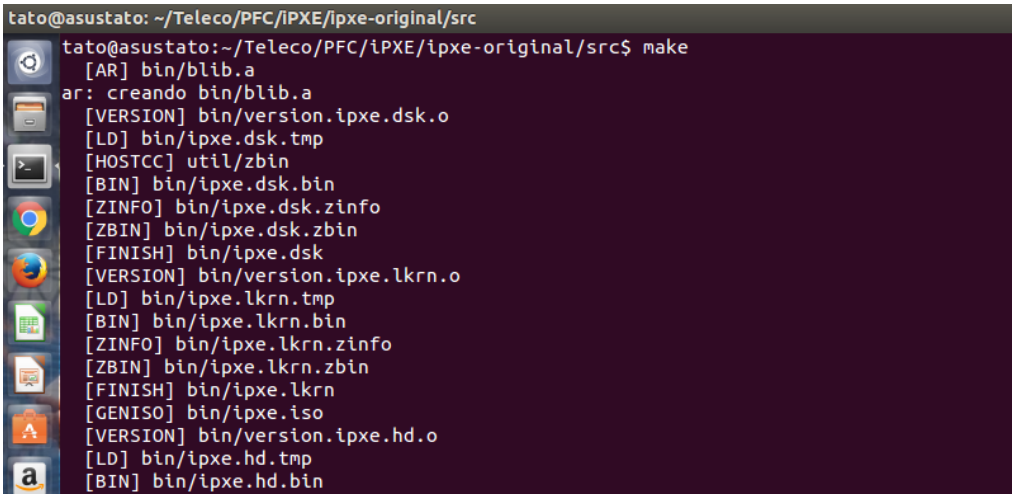


Ilustración 3- Compilando el código fuente haciendo uso de make

Como resultado del comando se generarán en el directorio *bin* las imágenes software del kernel iPXE en distintos formatos binarios. Algunas de ellas están pensadas para ser instaladas en la ROM de las tarjetas de red. Otras, como las de extensión .ISO o .LKRN, serán las que nos interesan dado que están

pensadas para residir en un dispositivo de almacenamiento como un disco duro, una memoria USB o un CD-ROM/DVD-ROM y ser arrancadas desde allí. El resultado de la compilación nos ofrece un listado de todos los formatos en los que ha sido compilado el kernel de iPXE.

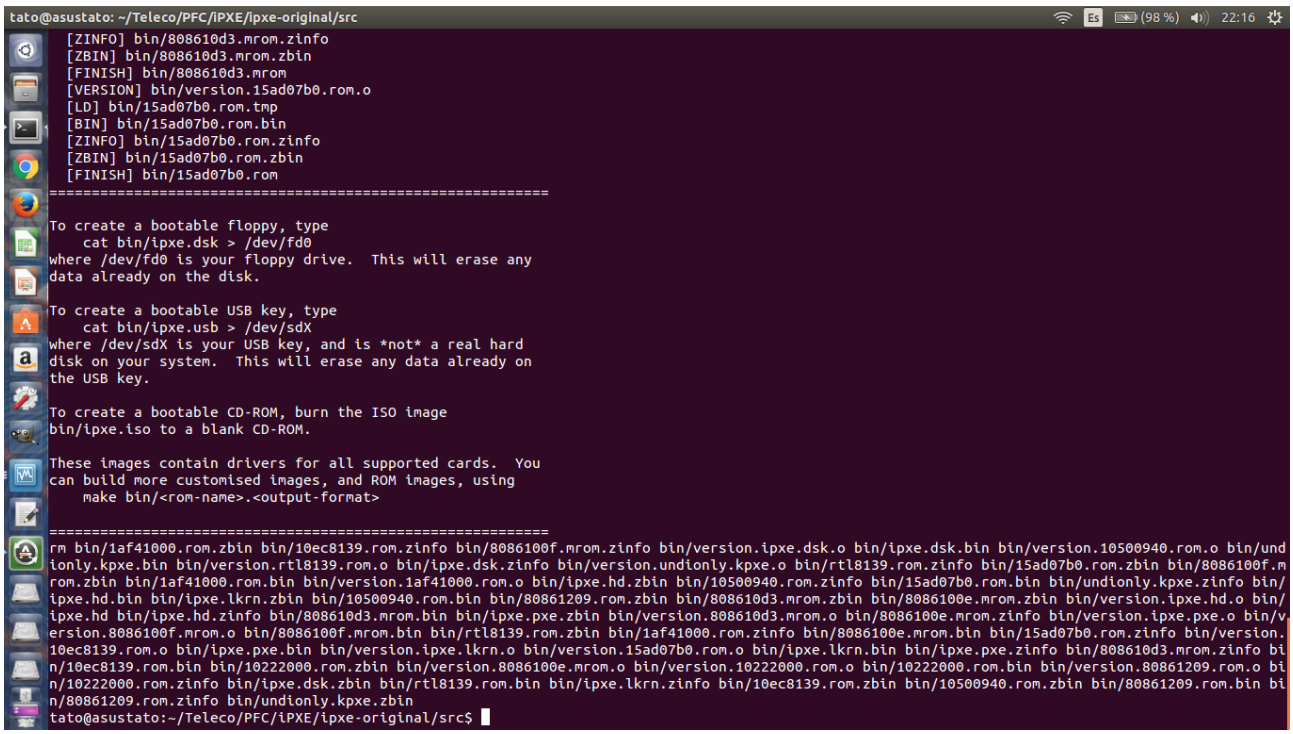


Ilustración 4 - Kernel de iPXE compilado

La siguiente imagen muestra el contenido de la carpeta *bin* en la que encontraremos las imágenes binarias que contienen el kernel de iPXE recién compilado. Se ha filtrado convenientemente el listado para que no muestre objetos intermedios ni ficheros temporales.

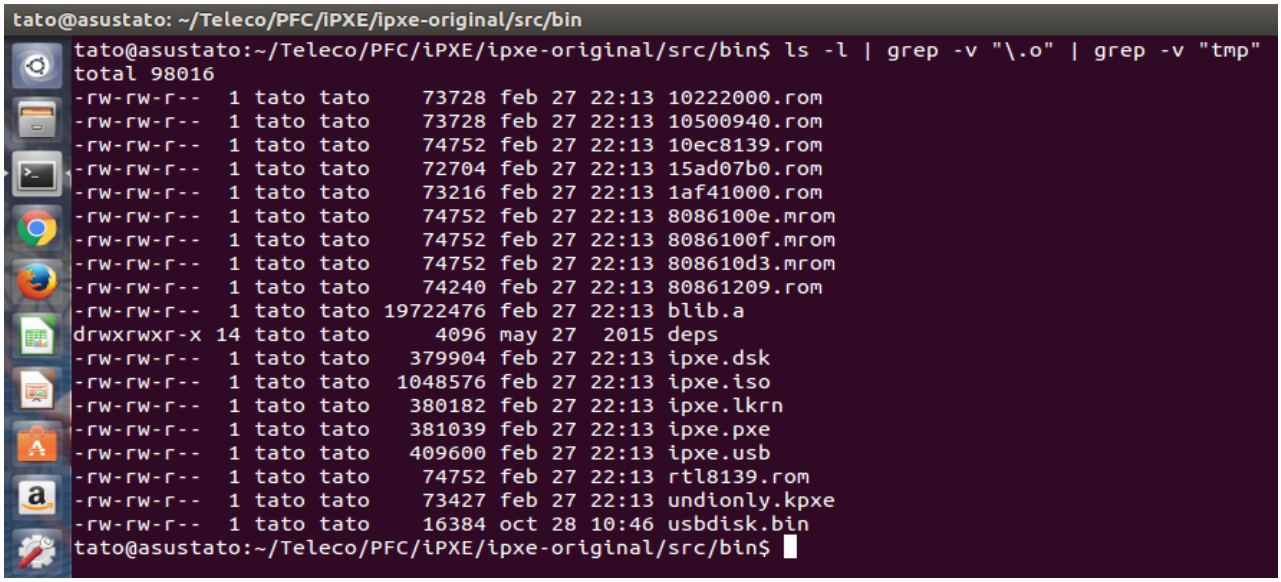


Ilustración 5- Imágenes binarias ejecutables resultado de la compilación

2.2.4.2. INSTALACIÓN DEL KERNEL iPXE COMPILADO

El último paso antes de poner en funcionamiento en el endpoint nuestro kernel de iPXE recién compilado es la instalación en el dispositivo de almacenamiento persistente del equipo, que para nuestros endpoints objeto de estudio será el disco duro. La instalación no consiste solamente en copiar el fichero obtenido sino que debemos dotar al equipo con un sistema que lo localice y ejecute antes de que se cargue el sistema operativo Windows. Nuevamente nos encontramos ante un problema de difícil solución puesto que no queremos eliminar el sistema operativo en disco del endpoint ni quedarnos sin la posibilidad de poder arrancarlo. Lo que necesitamos es que primeramente se cargue nuestro kernel de iPXE, realice las funciones necesarias, y posteriormente pase el testigo al sistema operativo instalado en disco para que sea éste el que termine de iniciar el equipo.

Es el momento de retomar el análisis que hicimos sobre la secuencia de arranque del endpoint.

Recordemos que en un sistema que cuenta con un disco duro rígido -en el que está instalado un sistema operativo complejo como Windows XP o Windows 7- el primer sector del disco se denomina Registro Maestro de Arranque (MBR) y que en estos 512 bytes se instala un código de arranque o bootstrap. Este código es muy sencillo, ocupa un máximo de 464 bytes y prácticamente su función se limita a localizar el programa *boot helper* adecuado (fichero NTLDR para Windows XP; fichero BOOTMGR para Windows 7) en alguna partición primaria para proseguir con la carga del sistema operativo.

Como se puede deducir, el código de arranque instalado en el MBR por los sistemas operativos Windows está preparado para cumplir con una única y sencilla función: localizar el programa *boot helper* concreto y cargarlo en memoria para que continúe con la carga de Windows. Por tanto, no es posible pedirle a este código de arranque que en lugar de cargar Windows cargue nuestro kernel de iPXE.

¿Cómo arrancar nuestro kernel de iPXE sin renunciar a cargar el sistema operativo Windows del endpoint?

La respuesta a esta pregunta es clara. No hay más alternativa que instalar un gestor de arranque y remplazar el código MBR original de Microsoft.

Un gestor de arranque es un programa que se instala (típicamente) en el Registro Maestro de Arranque del disco duro del PC y que permite arrancar múltiples sistemas operativos que residen en distintas particiones del disco duro. Algunos gestores de arranque pueden instalarse también al comienzo de una partición primaria, aunque no supone una diferencia significativa para nuestro propósito.

Si existe un gestor de arranque que goza de la máxima popularidad y prestigio en el mundo del PC ese es GRUB. Este conocido gestor de arranque ha evolucionado notablemente desde la primera versión y está presente en casi la totalidad de los sistemas basados en PC equipados con un sistema operativo de la familia Linux.

Pero, ¿qué pasa con los sistemas operativos Windows? También poseen la posibilidad de habilitar gestores de arranque multi-sistema operativo aunque en están pensados para arrancar sistemas operativos Windows principalmente.

GRUB sin embargo está pensado para arrancar sistemas operativos de distintas familias por lo que resulta adecuado para nuestro propósito de arrancar un kernel de iPXE, que es en definitiva, un pequeño sistema operativo con muchas limitaciones.

GRUB consta de dos componentes o etapas principales –hay quien habla de tres, pero no entraremos en dicho debate-. La primera de ellas es un código MBR que se instala en el primer sector del disco y que se extiende a los sectores sucesivos, siempre sin exceder los límites de la primera pista del disco. La segunda se trata de un boot helper (fichero *grldr*) que se instala en una partición primaria del disco.

Nótese que hemos enunciado que el código MBR de GRUB se instala en el primer sector del disco duro y en los sucesivos, mientras que el código MBR original de Windows se instala en el primer sector exclusivamente. Esto se debe a que en los 464 bytes del primer sector dedicados al código MBR no son suficientes para albergar la funcionalidad necesaria para la primera etapa de GRUB. Por tanto, se utilizarán los sectores sucesivos del disco, siempre sin exceder la primera pista. Este código instalado en los primeros sectores del disco duro estará encargado de localizar el fichero *grldr* buscándolo en las particiones primarias del disco. Una vez lo encuentre lo subirá a memoria y continuará con su ejecución.

GRUB4DOS es una implementación de GRUB pensada para su instalación en sistemas Windows. Aunque GRUB4DOS es un software considerado como obsoleto y no adecuado para equipos modernos, se adapta perfectamente a los endpoints bajo estudio y por ese motivo es la implementación seleccionada.

2.2.4.3. INSTALACIÓN DE GRUB4DOS EN EL MBR

La instalación de GRUB4DOS en el Registro Maestro de arranque más los sectores consecutivos es muy sencilla. Gracias a la aplicación *grubinst.exe* podremos instalar GRUB4DOS simplemente especificando el disco duro en el que debe ser instalado.

```
C:\> grubinst.exe (hd0)
```

Las siguientes tres imágenes muestran el Registro Maestro de arranque del disco duro tras la instalación de GRUB4DOS más los sectores segundo (sector 1) y tercero (sector 2).

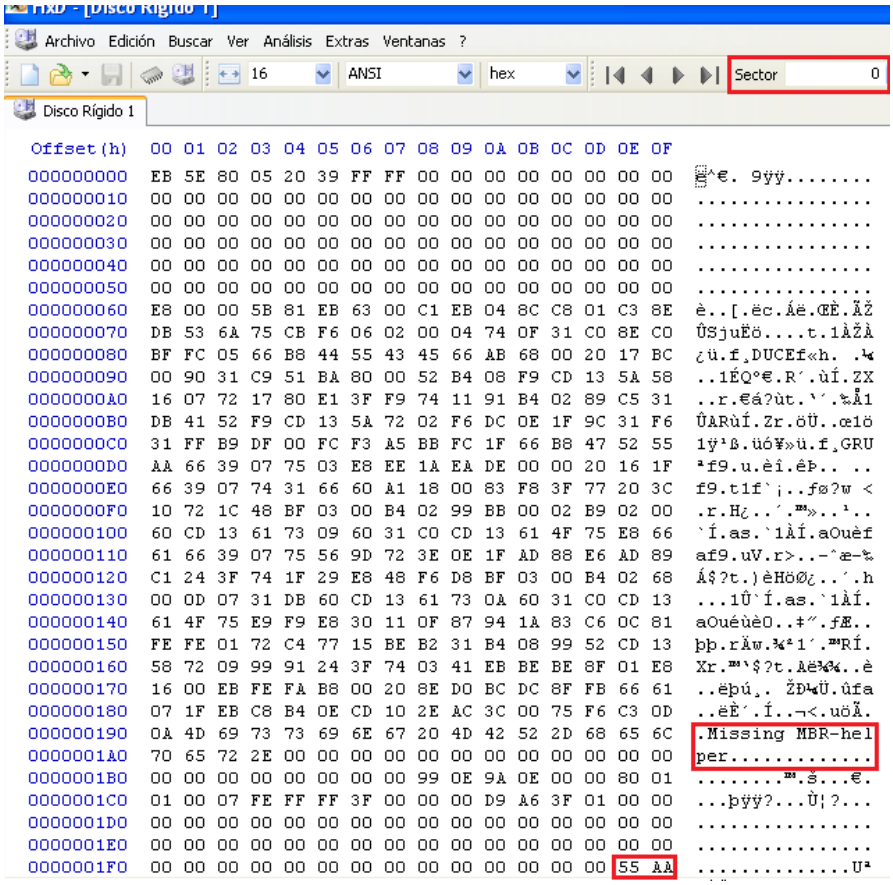


Ilustración 6- Código MBR de grub4dos (Sector 0)

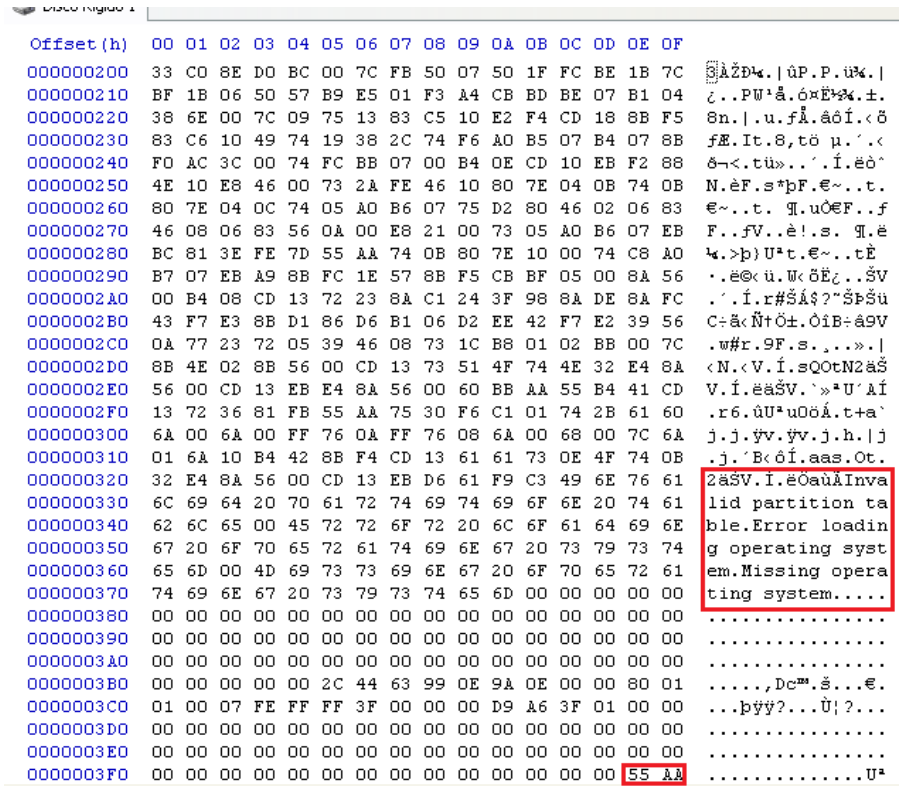


Ilustración 7- Código MBR de grub4dos (Sector 1)

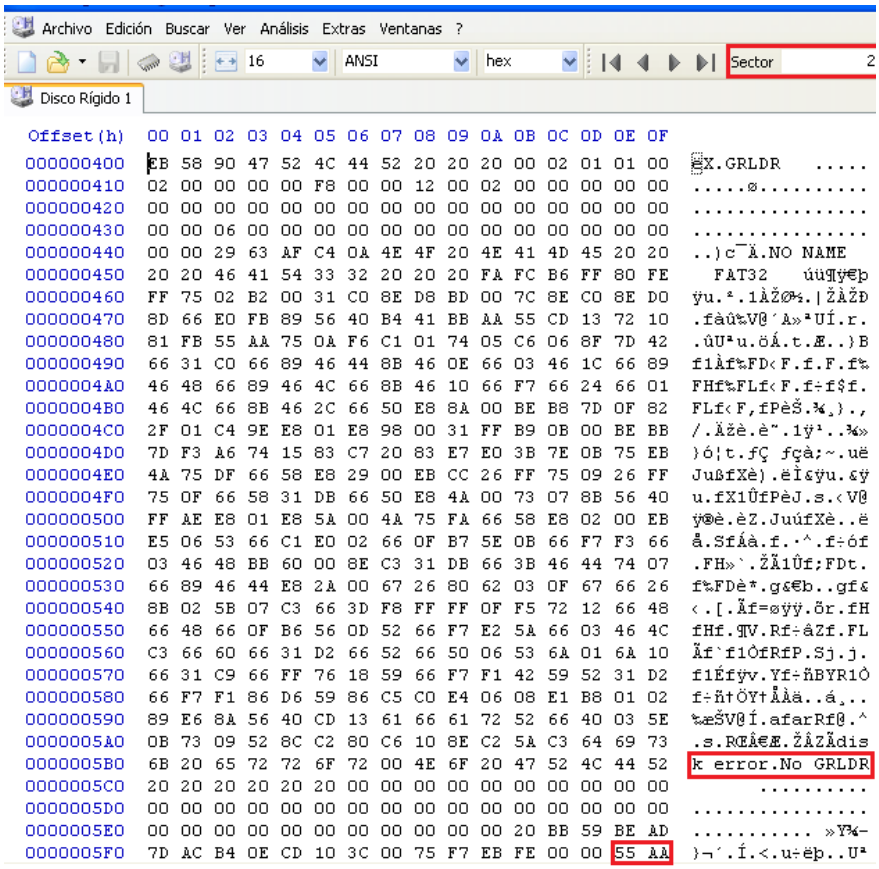


Ilustración 8- Código MBR de grub4dos (Sector 2)

Como se puede observar, el código MBR de grub4dos se extiende a los sectores contiguos al Registro Maestro de Arranque. Aunque es las imágenes solo se muestran los tres primeros sectores, el código MBR de grub4dos se extiende hasta el sector décimo sexto. Todos estos sectores cuentan con la bandera de código de arranque (55AAh).

Cabe destacar que en el segundo sector del disco (sector 1) encontramos el código MBR de Microsoft. Esto se debe a que la aplicación *grubinst.exe* realiza una copia de seguridad del MBR original en este sector cuando instala el código MBR de GRUB4DOS.

En la siguiente imagen se compara el primer sector de un disco en el que se encuentra instalador el cargador de estándar del sistema Windows XP con el primer sector de otro disco en el que se encuentra instalado GRUB4DOS.

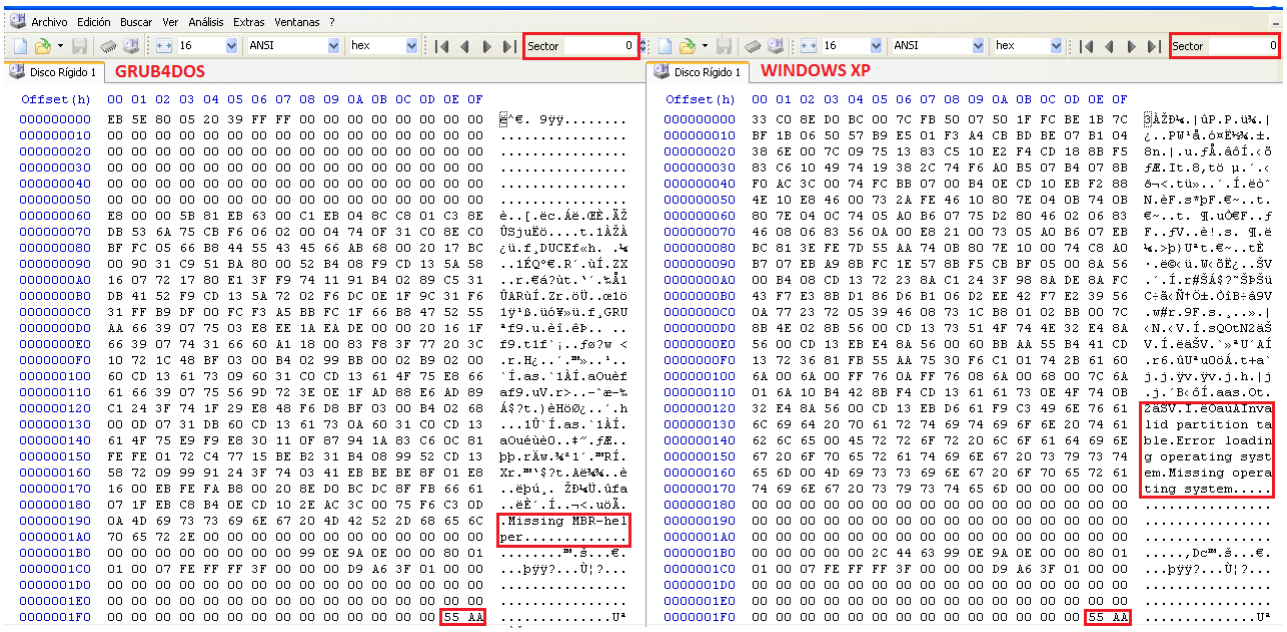


Ilustración 9- Comparativa entre el código MBR de GRUB4DOS y el de WINDOWS XP

La ejecución de grubinst.exe no se limitará a escribir el código MBR en el Registro Maestro de arranque sino que también escribirá los sectores sucesivos. Por otra parte, para poder dar por finalizada la instalación de GRUB4DOS es necesario dejar una copia de los ficheros *grldr* y *menu.lst* en la raíz de una partición primaria.

2.2.4.4. CONFIGURACIÓN DE GRUB4DOS PARA CARGAR iPXE

Como ya se ha indicado, una de las grandes ventajas de GRUB4DOS es que permite al usuario seleccionar el sistema operativo que se desea arrancar. Para ello, basta con editar el fichero *menu.lst*. Este fichero debe estar ubicado obligatoriamente en una partición primaria con formato FAT o NTFS.

A continuación se muestra el contenido del fichero *menu.lst* que nos permitirá elegir entre la carga del sistema operativo Windows XP o el kernel de iPXE.

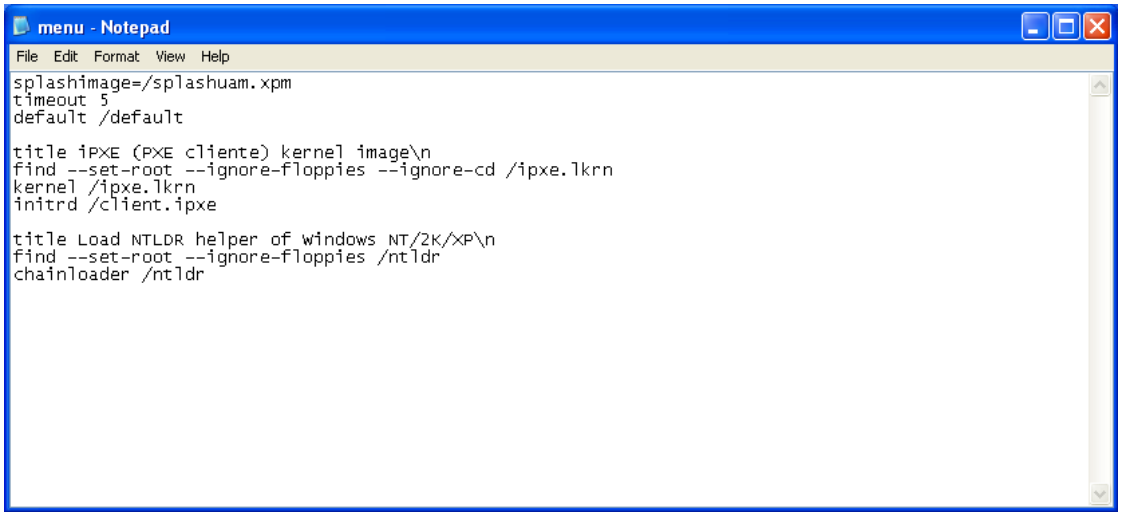


Ilustración 10 - Fichero de configuración del menú ofrecido por GRUB4DOS

La siguiente imagen corresponde al arranque de un endpoint con GRUB4DOS instalado y el fichero *menu.lst* cuyo contenido se acaba de mostrar en la imagen anterior.

Ilustración 11 - Menú de selección de sistema operativo de GRUB4DOS

Como se puede observar, al arrancar el endpoint se muestra un menú a partir del cual el usuario puede seleccionar qué sistema arrancar. En la imagen la elección se reduce a dos posibilidades: arrancar el kernel de iPXE o arrancar Windows XP. Obviamente, para entornos desatendidos habrá que especificar una opción por defecto de tal modo que transcurrido un tiempo prestablecido sea la elegida para iniciar el equipo. En nuestro caso será la de nuestro kernel de iPXE.

Llegados este punto solamente queda elegir la opción “iPXE” y comprobar que el equipo es capaz de arrancar el kernel.

A continuación se muestra una imagen del kernel de iPXE en ejecución. En la imagen se puede observar como el kernel obtiene una dirección IP para el endpoint mediante el protocolo DHCP.

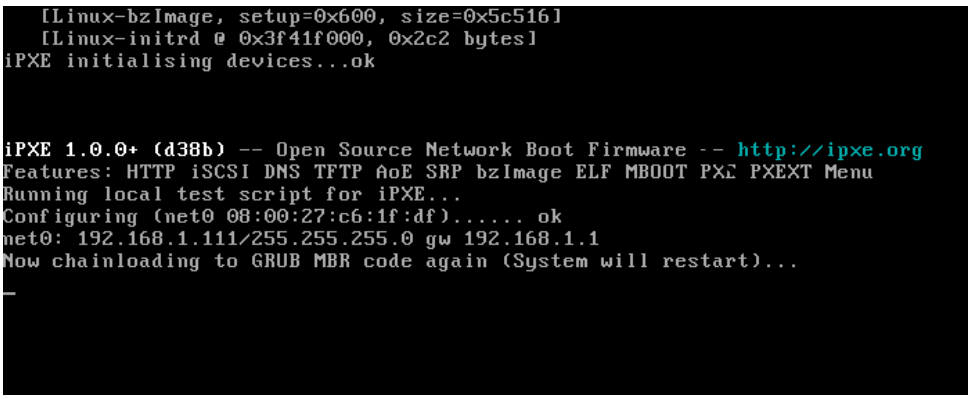


Ilustración 12 - Kernel de iPXE en ejecución

Como se puede ver en la imagen el kernel de iPXE obtiene además de una dirección IP (192.168.1.111), una máscara de red (255.255.255.0) y una puerta de enlace (192.168.1.1) del servidor DHCP que hay en la red en la que está conectado el endpoint de pruebas.

2.3. RESUMEN Y CONCLUSIONES

En este capítulo hemos abordado cómo conseguir soporte para comunicaciones en pre-boot. Veamos un pequeño resumen de lo conseguido hasta el momento así como las conclusiones más importantes de este capítulo.

Contamos con un kernel de iPXE moderno, que proporciona una implementación del cliente PXE básico atendiendo al estándar PXE más una serie de características avanzadas como soporte para protocolos de comunicación más complejos o la posibilidad de utilizar scripts que permiten cambiar el comportamiento del kernel durante su ejecución.

Además, se trata de un software de código abierto por lo que tenemos acceso a los archivos fuente y podremos modificarlos para adaptarlos a nuestras necesidades.

El kernel de iPXE soporta un gran número de tarjetas de red (aproximadamente 600 chipsets distintos) y su instalación o actualización en el endpoint es muy sencilla.

Por último, hemos identificado la necesidad de un gestor de arranque que nos permitirá dar entrada al kernel de iPXE durante la secuencia de arranque del endpoint para que sea dicho kernel el que se encargue de arrancar el equipo y no el cargador de Windows directamente.

En los próximos capítulos analizaremos la solución criptografía a emplear para securizar las comunicaciones. Será dicha solución criptográfica la que pasará a formar parte del kernel de iPXE. La integración entre la solución criptográfica diseñada y el kernel de iPXE la estudiaremos en el capítulo 0.

3. COMUNICACIONES SEGURAS

Es el momento de abordar el segundo de los dos grandes bloques de este proyecto: la securización de las comunicaciones. Aunque el término *securización* (al igual que el verbo *securizar*) no esté recogido en el Diccionario de la Real Academia Española, es considerado un tecnicismo, y por este motivo se prefiere el uso de este término al de *segurización* o *aseguramiento*. Obviamente, por securización de comunicaciones nos estamos refiriendo al hecho de hacer seguro el intercambio de información entre los extremos intervinientes.

A su vez, al hablar de securizar una comunicación, estamos incurriendo en una ambigüedad: ¿a qué nos referimos exactamente al emplear dicho término? Cuando hablamos de securizar una comunicación entre dos equipos interconectados por una red, no existe una definición exacta que desambigüe el significado de dicho verbo; es más, dependiendo de la aplicación y el contexto, el término *securizar* podría interpretarse de manera muy diferente.

En la materia que nos compete, y como se ha adelantado en el capítulo de introducción, consideraremos comunicación segura aquella que cumpla con los siguientes requisitos:

- Los extremos son autenticados, es decir, cada una de las partes intervinientes tiene la certeza de saber que está hablando con el interlocutor legítimo y no con un impostor.

- La información que se intercambia está debidamente cifrada, de modo que un observador del canal no puede leer el contenido de lo intercambiado.

Recordemos que a estos dos requisitos deseamos añadir un tercero por las características y necesidades de las redes de endpoints bajo estudio:

- No se hará uso de certificados digitales.

Con el anterior conjunto de tres requisitos es el momento de diseñar el sistema criptográfico (al que también nos referiremos como cripto-sistema para abreviar) adecuado que además debe ser aplicable a las comunicaciones en pre-boot descritas previamente.

Será precisamente el tercero de los requisitos el que nos lleve a basar nuestro cripto-sistema en lo que se conoce como Criptografía Basada en Identidad o por su acrónimo en inglés IBC (Identity Based Criptography), y esto a su vez en trabajar con unas estructuras algebraicas denominadas Curvas Elípticas. Pero no adelantemos acontecimientos y comencemos a construir la casa por los cimientos y no por el tejado.

A continuación dedicaremos algunas páginas a exponer e introducir conceptos básicos de criptografía que serán necesarios para una adecuada compresión del sistema criptográfico que se pretende construir. Lo que viene en adelante no pretende ser un manual de criptografía ni tampoco un resumen. Tampoco nos centraremos en la demostración matemática de los algoritmos utilizados. Existe un gran número de obras en la bibliografía especializada donde el lector interesado podrá encontrar todos estos detalles. Algunas de ellas han sido consultadas para la realización de este proyecto y están recogidas en el capítulo 8 de la presente memoria.

3.1. INTRODUCCIÓN A LA CRIPTOGRAFÍA MODERNA

La criptografía moderna se fundamenta en las mismas ideas básicas que la tradicional: la trasposición y la sustitución. Pero, mientras que la criptografía tradicional recurría al uso de algoritmos sencillos y claves muy largas, la moderna se fundamenta en hacer algoritmos de cifrado tan complicados y rebuscados que incluso cuando el criptoanalista obtiene cantidades enormes de texto cifrado no es capaz de entender nada, y por tanto no es capaz de descifrar los mensajes.

Podríamos decir que la criptografía moderna se divide en cifrado de clave privada y cifrado de clave pública.

En el cifrado de clave privada la clave privada y la clave pública son la misma, o bien se deriva de forma directa la una de la otra. A este tipo de cifrado también se le conoce como de clave simétrica.

Por su parte, en el cifrado de clave pública las claves de cifrado y de descifrado son independientes, no derivándose la una de la otra, por lo que puede hacerse pública la clave de cifrado siempre y cuando se mantenga en secreto la clave de descifrado, privada. Vamos a enumerar e ilustrar ahora el funcionamiento de algunos de los algoritmos más importantes o conocidos tanto de clave privada como de clave pública. Es posible que el lector que no tenga conocimientos básicos de criptografía encuentre alguna dificultad en la comprensión en este instante. Sin embargo, en los capítulos venideros vamos a realizar una introducción a la criptografía que va a definir los conceptos e ideas fundamentales que subyacen a esta disciplina y con los que el lector va a ser capaz de comprender los algoritmos presentados.

Los siguientes conocidos algoritmos corresponden a algoritmos de clave privada:

- **DES (Data Encryption Standard):** fue desarrollado por IBM en la década de los setenta. Utiliza claves de 56 bits, aunque deriva del algoritmo Lucifer, que utilizaba claves de 128 bits. La reducción de 128 a 56 bits fue propuesta por la NSA, lo que por aquel entonces causó una cierta controversia, ya que se pensó que la NSA había debilitado el algoritmo original intencionadamente. El algoritmo DES consta de 19 etapas y el texto a cifrar se descompone en bloques de 64 bits. En la actualidad está en desuso por ser considerado un algoritmo no seguro.

- **Triple DES:** consiste la utilización en cascada de tres cifradores tipo DES, como se ilustra en la figura.

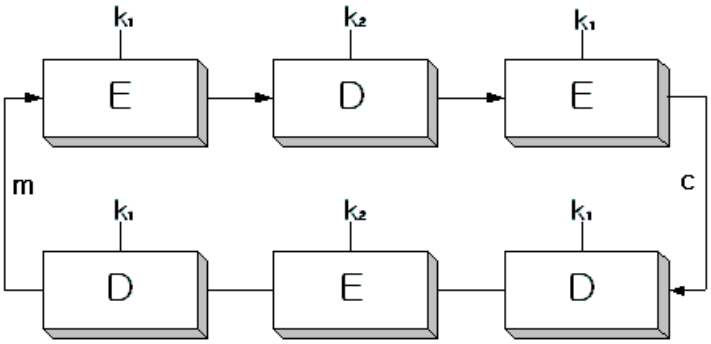


Ilustración 13 - Esquema de funcionamiento Tripe DES

Tanto pare el cifrado como para el descifrado se usan dos claves (k1 y k2) y tres etapas. A la hora de cifrar el texto normal m se aplica el algoritmo DES con clave k1 como primera etapa. En la segunda etapa el algoritmo DES se ejecuta en modo descifrado con la clave k2. Por último, en la tercera etapa se cifra nuevamente con la clave k1.

Al funcionar con dos claves de longitud 56 bits, se considera que triple DES utiliza claves de 112 bits, lo que se considera suficientemente seguro para una gran mayoría de aplicaciones comerciales.

Además, el hecho de cifrar con k1, descifrar con k2 y volver a cifrar con la misma clave k1 hace posible la compatibilidad entre sistemas que usan DES y sistemas que usan triple DES.

- **IDEA (International Data Encryption Algorythm):** fue diseñado en Zúrich y descrito por primera vez en 1991. Utiliza claves de 128 bits y fue propuesto como remplazo al algoritmo DES. Al igual que el algoritmo DES, alterna bloques de entrada de texto normal de 64 bits en una secuencia de iteraciones parametrizadas, aunque, dada la profunda alteración de los bits en cada iteración, basta con ocho iteraciones. La clave de 128 bits se utiliza para generar 52 subclaves de 16 bits. De las 52 subclaves se utilizan 6 en cada una de las 8 iteraciones, y las 4 restantes se emplean en la transformación final. El descifrado funciona del mismo modo. La única diferencia es que las 52 subclaves generadas a partir de la clave de 128 bits son diferentes.

- **AES (Advanced Encryption Standard):** sin duda alguna se trata de uno de los algoritmos de clave simétrica más utilizados en la actualidad. Surge en 1997 como resultado de un concurso propuesto por el NIST que tenía como objetivo seleccionar un nuevo algoritmo de cifrado que cumpliera una serie de requisitos. De todos los algoritmos presentados el que resultó elegido fue el Rijndael, que pasó a ser conocido posteriormente como AES. Su estandarización llevó cinco años, por lo que no fue anunciado hasta 2001.

  AES admite claves de 128, 192 y 256 bits. Opera sobre matrices de 4x4 bytes (32x32 bits). Para una clave de 128 bits AES ejecuta 10 iteraciones sobre dichas matrices. Para una clave de 192 bits lo hará en 12 iteraciones, y finalmente, para claves de 256 ejecutará 14 iteraciones.

  AES es considerado un algoritmo robusto por la NSA. Ya en su versión de 128 bits se considera adecuado para cifrar información secreta. El uso de claves de 192 y 256 bits se considera reservado para cifrar información de alto secreto. A pesar de ello, existen algunos algoritmos teóricos que ponen en tela de juicio la seguridad de AES, aunque todos ellos generan controversia entre los criptoanalistas.

Todos estos algoritmos de clave privada adolecen del mismo problema: la dificultad de la distribución y preservación de las claves. Si alguien malintencionado se hace con las claves el sistema criptográfico, no vale para nada por muy robustos que sean los algoritmos en los que se basa.

Para solucionar este problema aparecieron los cripto-sistemas de clave pública, los cuales fueron propuestos para cumplir con los siguientes requisitos:

1) D(E(m)) = m. Siendo D el algoritmo de descifrado, E el algoritmo de cifrado y m el texto normal a cifrar.
2) No se puede deducir D de E.
3) No se puede descifrar E(m) mediante un ataque de texto normal seleccionado.

Una vez construidos los algoritmos E y D, la persona o sistema R que quiere recibir mensajes de forma segura hace públicos E, D y la clave de cifrado (E_R), pero mantiene en secreto la clave de descifrado (D_R).

El sistema o persona S que desea enviar información de forma segura a R, conoce E, D y E_R (ya que son públicos), además de lógicamente su propia clave privada D_S. Al igual que R, el emisor S también hará pública su clave de cifrado, E_S.

S coge el mensaje a enviar, m, y calcula $E_R(m)$ y se lo envía a R. Cuando R recibe el mensaje utiliza su clave privada calculando $D_R(E_R(m))$, lo que resulta en el mensaje original m.

Si alguien intercepta el mensaje $E_R(m)$ no puede descifrarlo porque los algoritmos son robustos y la clave de descifrado no puede ser derivada a partir de la clave de cifrado, que es pública.

A continuación se exponen dos de algoritmos más conocidos de cifrado de clave pública. Si el lector no tiene conocimientos de criptografía, puede que no le resulte sencillo seguir algunos de los detalles expuestos para los algoritmos. En ese caso, se recomienda al lector volver a este apartado una vez que haya finalizado la lectura de todo el capítulo 3, puesto que durante el desarrollo del capítulo se va a realizar una introducción a la criptografía. Por el momento, presentemos los algoritmos:

- **DSA (Digital Signature Algorithm):** como su propio nombre indica, se trata de un algoritmo para producir y verificar firmas digitales. No puede ser utilizado para cifrar/descifrar información. Fue propuesto por el NIST en 1991 y adoptado como estándar FIPS en 1993. La longitud de las claves es de 1024 bits (FIPS 186-2 así lo define), aunque teóricamente sería posible utilizar claves de longitud mayor, como se indica en FIPS 186-3. No entraremos en más detalles sobre este algoritmo, ya que como se ha mencionado no se trata de un algoritmo de cifrado, sino de firma.

- **RSA (Rivest Shamir Adleman):** los investigadores del MIT cuyas iniciales dan nombre al algoritmo lo diseñaron para satisfacer los tres requisitos que hemos mencionado anteriormente para los sistemas de cifrado de clave pública. RSA es sin duda uno de los algoritmos más extendidos actualmente en multitud de aplicaciones informáticas que requieren comunicaciones seguras. Por la importancia y relevancia de este algoritmo veremos cómo funciona.

  Lo primero de todo es ejecutar una fase de inicialización, que consiste en los siguientes cuatro pasos:

  - En primer lugar se seleccionan dos números primos muy grandes: p y q.

  - Después se calculan los siguientes dos productos: n = p x q y z = (p-1) x (q-1).

  - Después se selecciona otro número d que no tiene factores comunes con z.

  - Por último se calcula e tal que el producto e x d es congruente con 1 módulo z, es decir, e x d ≡ 1 mod z.

Una vez finalizada la fase de inicialización, el cifrado del texto normal m tiene lugar del siguiente modo:

- Se divide el texto normal m en bloques de P bits tal que 0 < P < n.
- Para cifrar P calculamos $P^e \bmod n$ produciendo C.
- Para descifrar C y de este modo recuperar P calcularemos $C^d \bmod n$.

Por tanto para la función de cifrado es necesario conocer e y n, mientras que para la función de descifrado necesitaremos d y n. De aquí concluimos que la clave pública es el par (e,n) y la clave privada el par (d,n).

Al ser n un dato público la seguridad de RSA reside en la dificultad de factorizarlo. Si alguien factoriza n obtendría p y q, y con ellos, z. Conociendo z y e (que también es público) puede encontrar d utilizando el Algoritmo de Euclides.

Aunque se sigue avanzando en la factorización de números grandes, se considera aún un problema excesivamente difícil de resolver para números grandes. Los autores de RSA afirman que la factorización de un número de 200 dígitos requeriría actualmente más de un millón de años de tiempo de cómputo.

Hasta hace unos pocos años se utilizaban claves de 1024 bits aunque en la actualidad se recomienda el uso de claves de 2048 bits. Esta longitud se estima que será suficiente hasta el año 2030. En el apéndice E se ilustra una tabla con la fecha de validez estimada de las claves en función de su longitud (en bits).

En la actualidad RSA es el algoritmo más extendido para la firma de los certificados digitales que utilizan los servidores de los sitios web que visitamos con nuestros navegadores.

En este proyecto no se pretende poner en tela de juicio la seguridad o la conveniencia de RSA; simplemente nos centraremos en otros algoritmos diferentes que, para proporcionar la misma seguridad que RSA, utilizan claves de menor tamaño, lo que será conveniente para nuestros endpoints, limitados en capacidad de computo.

3.2. EL PROBLEMA DEL LOGARTIMO DISCRETO EN Z_q

Acabamos de ver cómo la fortaleza del algoritmo RSA reside en la dificultad para factorizar números grandes. Existen otros problemas matemáticos que por ser de difícil solución resultan convenientes para construir sistemas criptográficos seguros. Otro problema matemático de difícil solución que se emplea para construir cripto-sistemas es el conocido como Problema del Logaritmo Discreto sobre campos finitos. Antes de enunciar en qué consiste el problema exactamente, necesitamos ilustrar una serie de conceptos básicos. Estos conceptos básicos no se aplican exclusivamente al Problema del Logaritmo Discreto, pero sí son necesarios para entender dicho problema.

3.2.1. ARITMÉTICA MODULAR

La aritmética modular se construye a partir de la relación de congruencia de los números enteros. Dos números enteros a y b son congruentes módulo un tercer número n si ambos producen el mismo resto al ser divididos por n. La relación de congruencia se expresa típicamente así:

$$a \equiv b \bmod n$$

Por ejemplo los número 23 y 9 son congruentes módulo 7 porque para ambos el resto de la división entera entre 7 produce el valor 2.

$$23 \equiv 9 \bmod 7$$

Existe un número infinito de números que son congruentes con otro módulo un tercero, pero lo habitual es trabajar con el entero positivo que sea inferior al módulo. En el ejemplo anterior nos quedaríamos con el número 2, que es positivo e inferior a 7.

Al establecer un valor concreto para el módulo, estamos acotando el número de enteros posibles. Si definimos que el módulo es 7, únicamente serán posibles los restos 0, 1, 2, 3, 4, 5 y 6. De este modo en la aritmética modular los números enteros se pliegan o enrollan cada vez que se alcanza un determinado valor, el módulo.

El ejemplo más claro de la vida cotidiana donde encontramos aritmética modular es un reloj de agujas. La aguja de las horas marca desde las 0 horas (o las 12) hasta las 11. Si tenemos un reloj que marca las 5 y transcurren 10 horas, el rejo no marcará las 15, sino las 3.

Otro ejemplo es el odómetro de los coches antiguos. Típicamente consta de cinco dígitos que van llevando la cuenta de los kilómetros recorridos por el vehículo, por lo que permiten registrar hasta 99.999 km. Cuando se supera ese número el odómetro vuelve a indicar 00.000 km, lo que no deja de ser un hecho simpático que rejuvenece nuestro vehículo sin necesidad de pasar por taller. Si a nuestro viejo automóvil de 95.000 km le sometemos a un recorrido de 8.000 km, el odómetro pasará a indicar 3.000 km.

En aritmética modular no solo podemos sumar enteros sino también restar, multiplicar y dividir (multiplicar por el inverso).

3.2.2. CAMPOS FINITOS DE ENTEROS MÓDULO UN NÚMERO PRIMO

Nos referimos al campo finito Z_q como el conjunto de números enteros que va desde el 0 hasta el q-1 donde q es un número primo. Para este campo consideramos las operaciones de suma, resta, multiplicación y división, todas ellas módulo q.

A continuación se explica brevemente cómo se realiza cada una de dichas operaciones en Z_q.

- **Suma de a y b mod q:**

Se suma a y b y se busca el menor número entero positivo que es congruente con el resultado obtenido módulo q.

Ejemplo: sumar 25 + 13 en Z_{11}

i. 25 + 13 = 38
ii. 38 es congruente con 5 módulo 11.
iii. Por tanto, 25 + 13 = 5 en Z_{11}

- **Resta de a y b mod q:**

Se resta b de a y se busca el menor número entero positivo que es congruente con el resultado obtenido módulo q.

Ejemplo: calcular 7 − 15 en Z_{13}

i. 7 − 11 = -4
ii. -4 es congruente con 9 módulo 13.
iii. Por tanto, 7 − 11 = 9 en Z_{13}

- **Multiplicación de a por b mod q:**

Se multiplican a y b y se busca el menor número entero positivo que es congruente con el resultado obtenido módulo q.

Ejemplo: calcular 5 x 7 en Z_{17}

i. 5 x 7 = 35
ii. 35 es congruente con 1 módulo 17
iii. Por tanto, 5 x 7 = 1 en Z_{17}

- **División de a entre b mod q:**

En el caso de la división no es tan inmediato obtener el resultado módulo q. En realidad transformamos la división por una operación de multiplicación por el inverso. El inverso de un número p es aquel número w tal que p x w = 1 módulo q.

En el ejemplo de la multiplicación de 5 x 7 en Z_{17} obtenemos que el resultado es el número 1. Por tanto podemos concluir que 7 es el inverso de 5 en $Z_{17.}$ En consecuencia, para dividir por 5 en Z_{17} basta con multiplicar por 7.

Ejemplo: calcular 12 / 5 en Z_{17}

i. 12 / 5 se transforma en 12 x 7 en Z_{17}
ii. 12 x 7 = 84
iii. 84 es congruente con 16 módulo 17
iv. Por tanto, 12 / 5 = 16 en Z_{17}

Introduciendo números complejos obtenemos una extensión del campo Z_q que da lugar al campo Z_q^2. Para que este campo Z_q^2 nos resulte de utilidad, necesitamos tener disponibles las mismas cuatro operaciones que en Z_q, es decir, suma, resta, multiplicación y división.

Recordemos que los números complejos son de la forma a + ib, donde i es la unidad imaginaria. Para el campo Z_q^2 elegiremos que a y b sean números enteros, dado que queremos seguir trabajando con aritmética modular.

La operaciones de suma, resta, multiplicación y división definidas para Z_q siguen aplicándose en Z_q^2. Simplemente hay que tener en cuenta que $i^2 = -1$ y que para obtener el inverso de un número complejo hemos de multiplicar el numerador y el denominador por el complejo conjugado del número.

Una vez introducidos los conceptos básicos de la aritmética modular y los campos finitos Z_q y Z_q^2 es el momento de definir el Problema del Logaritmo Discreto sobre estos campos:

Dados dos números g y v pertenecientes a Z_q (o a Z_q^2), el Problema del Logaritmo Discreto también conocido por las siglas DLP (Discrete Logarithm Problem), consiste en encontrar otro número x tal que $g^x = v$

g^x se calcula multiplicando g por sí mismo x veces.

Veamos un sencillo ejemplo para entender de qué estamos hablando y qué es lo que subyace detrás de este problema matemático:

Consideremos que estamos trabajando en Z_{23}. Supongamos que conocemos g = 3 y v = 4. El Problema del Logaritmo Discreto consiste en encontrar un número x tal que $3^x = 4 \pmod{23}$. Este problema es difícil de resolver. No existe un algoritmo eficiente que proporcione una solución. Para este ejemplo el algoritmo de ir eligiendo valores para x y comprobar el resultado es adecuado. De este modo, cuando probáramos con x = 3 encontraríamos que $3^3 = 4\ mod\ 23$, luego x = 3 es la solución a este problema. Como es lógico, este método de probar todos los posibles valores de x solamente tiene sentido cuando q es un número pequeño (en este ejemplo, q = 23). Para valores grandes de q el número de valores de x a probar resulta tan grande que no resulta posible llevarlo a la práctica.

El hecho de que este problema sea de difícil solución, al igual que sucede con la factorización de números muy grandes expuesta anteriormente durante la presentación del algoritmo RSA, posibilita la construcción de sistemas criptográficos. Veamos un ejemplo de sistema criptográfico que hace uso del Problema del Logaritmo Discreto: se trata del conocido protocolo *Diffie-Hellman* para el establecimiento de claves entre pares que no han tenido contacto previo.

3.2.3. PROTOCOLO DIFFIE-HELLMAN

Supongamos que José y Julián quieren intercambiar cierta información de manera confidencial. Lo primero que deben hacer es acordar una clave que posteriormente les permita cifrar la información que van a trasmitir. Típicamente acordarán una única clave para ser utilizada en un algoritmo de cifrado simétrico de los que expusimos anteriormente (Triple DES, AES, IDEA, etc.)

Pero ¿cómo pueden acordar dicha clave si el canal del que disponen para intercambiar información es público y todo el mundo puede observar los datos que intercambian? Esto es lo que va a facilitar el protocolo Diffie-Hellman precisamente.

Veamos cómo funciona Diffie-Hellman para el establecimiento de la clave:

- José selecciona un número primo q y otros dos números aleatorios cualesquiera g y x. Calcula g^x módulo q y envía a Julián a través del canal inseguro la siguiente terna: q, g y g^x

- Julián recibe la terna enviada por José, selecciona otro número aleatorio z y calcula g^z. A continuación envía de vuelta a José g^z.

- Cuando Julián recibe el mensaje de José, ambos pueden calcular la misma clave $g^{zx} = g^{xz}$ (módulo q).

Lo interesante de este intercambio es que Julián y José pueden calcular la clave, y nadie más puede hacerlo aun habiendo observado todo el intercambio. Esto es posible porque el Problema del Logaritmo Discreto es difícil de resolver.

Sin embargo, el protocolo *Diffie-Hellman* para el establecimiento de claves es vulnerable a los denominados ataques de *man-in-the-middle* (hombre en el medio).

Imaginemos ahora que la información que Julián y José quieren intercambiar también interesa a Igor, quien además está en disposición de escuchar el canal por el que tiene lugar la conversación.

En el esquema anterior, cuando José inicia la conversación y envía la información pública a Julián, Igor intercepta el mensaje y responde a José con su propia información pública. A su vez Igor contacta con Julián, a quien facilita su información pública como si se tratara de la de José. De este modo, Igor, que está en el medio de la comunicación, establece una clave para hablar con José y otra distinta para hablar con Julián. Por su parte, Julián y José son ajenos a lo que está sucediendo y piensan que están hablando el uno con el otro.

Cada información que José envía a Julián es recibida en primer lugar por Igor, quien la descifra, lee, y posiblemente modifica. Después la cifra con la clave de sesión acordada con Julián y se la entrega.

Este problema surge porque el protocolo Diffie-Hellman no autentica a los extremos participantes de la comunicación.

Vulnerabilidades del tipo *man-in-the-middle* aparte, veamos rápidamente cómo de costoso es romper Diffie-Hellman, es decir, cómo de costoso es calcular la clave g^{zx} conociendo g^z o bien g^x. Intuitivamente observamos que construir la clave final está directamente relacionado con el número de posibilidades que tendríamos que probar para x (o para z). En el ejemplo con el que ilustrábamos el Problema del Logaritmo Discreto, el valor utilizado para q era 23. De este modo g^x solo podría tomar 23 valores distintos y, por tanto, tras 23 intentos como máximo encontraríamos la clave. El procedimiento de probar con todas la posibles salidas de g^x tiene un coste exponencial, es decir, que el número de intentos a realizar crece exponencialmente con q.

Para finalizar con el análisis del Problema del Logaritmo Discreto, presentaremos brevemente el algoritmo *index-calculus*. Este algoritmo fue propuesto para resolver el DLP con un coste sub-exponencial. Se trata del mejor algoritmo conocido hasta la fecha y, aunque reduce significativamente la resolución del DLP, este sigue siendo inabordable para valores muy grandes de q. La aparición de este algoritmo forzó el uso de valores de q más grandes para mantener los mismos niveles de seguridad. Utilizar valores mayores de q implica que las operaciones aritméticas requieren mayor tiempo de cómputo. Como consecuencia de la puesta en escena del algoritmo *index-calculus,* los matemáticos comenzaron a buscar nuevos campos que fueran inmunes a él.

3.2.4. CURVAS ELÍPTICAS

Al término de la sección anterior presentábamos el algoritmo *index-calculus* que simplificaba la solución del Problema del Logaritmo Discreto y la motivación que suponía para los matemáticos a la hora de encontrar nuevos campos en los que este nuevo enemigo no fuera aplicable. Entre otros resultados, la búsqueda de los matemáticos resultó en la aparición de un nuevo tipo de criptografía de clave pública

basada en la estructura algebraica de las curvas elípticas. Este nuevo tipo de criptografía recibe el nombre, por motivos obvios, de Criptografía de Curva Elíptica, también conocida por sus siglas en inglés, ECC (Elliptic Curve Criptography).

Veamos qué son dichas curvas elípticas y cómo pueden usarse para construir sistemas criptográficos.

En primer lugar, una curva elíptica E no es más que un conjunto de puntos que satisfacen la siguiente ecuación:

$$E: y^2 = x^3 + ax + b$$

donde cada punto queda definido por sus coordenadas (x,y).

Distintos valores de a y b dan lugar a distintas curvas elípticas.

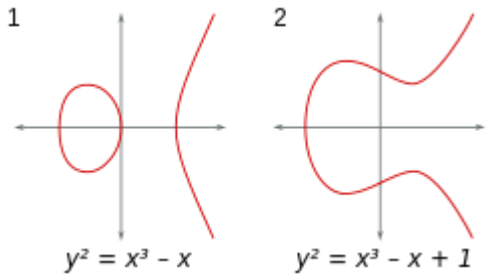


Ilustración 14 - Ejemplo de curvas elípticas

Diremos que un punto de coordenadas (x,y) pertenece a la curva elíptica si satisface la ecuación anterior una vez fijados los valores de a y de b.

Con vistas a construir cripto-sistemas basados en curvas elípticas, al igual que sucedía con los campos Z_q y Z_q^2, necesitaremos definir una serie de operaciones que nos permitan operar con los puntos de la curva. En esta ocasión no necesitaremos definir las operaciones de suma, resta, multiplicación y división, sino que será suficiente con definir la suma, el doblaje y la multiplicación de un punto por un número. La primera de ellas nos permite añadir un punto de la curva a otro y obtener un tercer punto que también pertenece a la curva. La operación de doblaje nos permite añadir un punto a sí mismo obteniendo de nuevo otro punto de la curva. Veamos cómo se realiza cada una de estas operaciones. Por último, la operación de multiplicación escalar nos permite multiplicar un número por un punto y obtener otro punto de la curva.

Suma de P y Q:

Sean P y Q dos puntos de la curva elíptica definidos por sus respectivas coordenadas (x_P,y_P) y (x_Q,y_Q), la suma de P y Q, representada como P + Q da lugar a un tercer punto W de coordenadas (x_W,y_W), donde las coordenadas de W se obtienen del siguiente modo:

$$W = P + Q$$

$$s = (y_P - y_Q) / (x_P - x_Q)$$

$$x_W = s^2 - x_P - x_Q$$

$y_W = -y_P + s(x_P - x_W)$

donde s es la pendiente de la recta que forman P y Q.

Doblaje de P:

Esta operación en inglés se conoce como *Doubling*

Sea P un punto de la curva elíptica definido por sus coordenadas (x_P, y_P), el doblaje de P, representado como 2P da lugar a un nuevo punto W cuyas coordenadas (x_W, y_W) se obtienen de la siguiente manera:

$W = 2P$

$s = (3x_P^2 + a) / (2y_P)$

$x_W = s^2 - 2x_P$

$y_W = -y_P + s(x_P - x_W)$

donde a es uno de los dos coeficientes que definen la curva.

Multiplicación de m por P (multiplicación escalar):

Sean P un punto de la curva y m un número (veremos más adelante qué restricciones tenemos para m), la multiplicación de mP da lugar a un nuevo punto W que también pertenece a la curva y cuyas coordenadas (x_W, y_W) se pueden obtener a partir de las operaciones de suma y doblaje.

Por ejemplo para calcular la multiplicación de m=7 por el punto P, combinaremos sumas y doblajes del siguiente modo:

$W = 7P = 2(2P+P)+P$

No tenemos necesidad de definir una operación de resta como tal, pero sí estaremos interesados en poder sustraer un punto de otro. Para ello necesitamos poder definir el negativo de un punto el cual sumado con el punto, resulte en el elemento *identidad.* Veamos pues la definición del negativo de un punto y del elemento o punto *identidad*.

Negativo de un punto:

Sea P un punto de la curva elíptica definido por sus coordenadas (x_P, y_P), el negativo de P, representado por –P, es otro punto de la curva elíptica cuyas coordenadas son $(x_P, -y_P)$.

El negativo de P es por tanto el simétrico de P respecto del eje de abscisas.

Elemento identidad:

También se conoce al elemento identidad como Punto en el Infinito y se suele presentar por la letra O. El elemento identidad es aquel que cumple las siguientes condiciones (siendo P un punto de la curva elíptica):

i. $P + O = P$
ii. $P + (-P) = O$

Tras todas estas definiciones de puntos y operaciones sobre puntos, falta exponer una condición fundamental para que podamos utilizar las curvas elípticas como base de sistemas criptográficos: las coordenadas de los puntos de la curvas deben pertenecer a algún campo finito. Nos centraremos únicamente en los campos Z_q y Z_q^2 vistos anteriormente. Existen otros campos finitos posibles, pero no los abordaremos en este proyecto. El campo finito seleccionado recibe el nombre de *campo subyacente*. La selección un campo finito lleva intrínseca la selección de un número primo grande q. Tanto las coordenadas de los puntos de la curva como los números implicados en las operación, e incluso los parámetros que definen la curva (a y b), pertenecerán al conjunto de los números enteros de 0 hasta q-1 y adicionalmente la unidad imaginaria i.

Acabamos de ver lo que son las curvas elípticas, pero ¿por qué son útiles de cara a construir sistemas criptográficos? La respuesta es que nos van a ayudar a plantear problemas de difícil solución y a obtener funciones que trabajan en un solo sentido. Como ya se ha expuesto, estas son las bases para construir cripto-sistemas seguros.

Pero antes de meternos en harina, necesitamos definir otros parámetros de las curvas elípticas.

Orden de la curva elíptica:

Se define el orden de una curva elíptica, representado como n, como el número de puntos de la curva.

El número de puntos de una curva elíptica se puede calcular a partir del número de puntos del campo finito subyacente. Por ejemplo, el teorema de Hasse relaciona el número de puntos del campo subyacente con el número de puntos de la curva según la siguiente relación:

$$|N - (q + 1)| \leq 2\sqrt{q}$$

Pero por simplificar las cosas podemos asumir que el orden de una curva elíptica sobre Z_q y Z_q^2 es del orden de magnitud de q y q^2 respectivamente, y por tanto podemos asumir que el número de puntos de una curva elíptica es aproximadamente al número de elementos del campo subyacente.

Orden de un punto en una curva elíptica:

Hemos definido unas líneas más arriba la multiplicación de un número por un punto o multiplicación escalar. El resultado de esta operación es un nuevo punto de la curva. Sin embargo, fijado un punto P, ¿cuántos posibles resultados hay para mP? Resulta que las curvas elípticas tienen la propiedad de que, a partir de un determinado valor de m, los resultados comienzan a repetirse dando lugar a ciclos. A este valor r que hace que rP = O se le conoce como orden del punto, y lo que viene a indicar es que solamente existen r puntos como resultado de mP. Esos r puntos forman un subgrupo cíclico de la curva. Al punto P se le conoce como punto generador del subgrupo.

Para una mejor compresión, veamos un ejemplo. Consideremos que hemos seleccionado una curva elíptica y un punto P que pertenece a dicha curva. La curva elíptica está definida sobre Z_q con q = 13. Si

multiplicamos m por P, comenzando por m = 0 y aumentándolo de unidad en unidad, obtenemos los siguientes puntos:

0P = O; 1P = P; 2P = Q; 3P = W; 4P = T; 5P = O;

Q, W y T son puntos de la curva distintos de P. O es el punto en el infinito. Cuando llegamos a m = 5, volvemos a obtener el punto en el infinito, O.

En este ejemplo, r=5 y por tanto diremos que P es un punto (generador) de orden 5 que da lugar al subgrupo formado por los puntos O, P, Q, W y T.

Para finalizar con el ejemplo, simplemente mencionaremos que si siguiéramos aumentando el valor de m por encima de r obtendríamos los siguientes resultados:

6P = P; 7P = Q; 8P = W; 9P = T; 10P = O;

Para cada punto P de la curva elíptica siempre existe un valor r tal que rP = O. Además, el valor de r puede ser diferente para cada punto seleccionado de la curva.

Para finalizar expondremos que, para todo valor primo r que divide al número total de puntos de la curva n, siempre existe un punto P de orden r.

3.2.5. EL PROBLEMA DEL LOGARITMO DISCRETO EN CURVAS ELÍPTICAS

En apartados anteriores hemos visto cómo el Problema del Logaritmo Discreto sobre el campo Z_q resulta adecuado para la construcción de sistemas criptográficos, puesto que es difícil de resolver.

El Problema del Logaritmo Discreto se puede definir también cuando trabajamos con curvas elípticas. Cuando este es el caso, se le conoce por sus siglas en inglés ECDLP (Elliptic Curve Discrete Logarithm Problem) y su definición es la siguiente:

Sean P y Q puntos pertenecientes a la curva elíptica, encontrar x tal que xP = Q

Recordemos que xP es la multiplicación de un número (x) por un punto (P) y ya hemos visto cómo se computa. Por otra parte, dos puntos de una curva cumplen la condición de igualdad cuando sus coordenadas son iguales dos a dos, es decir, cuando la coordenada x del primer punto es igual a la coordenada x del segundo punto y cuando la coordenada y del primer punto es igual a la coordenada y del segundo punto.

Recordemos también que cuando hablamos de curvas elípticas hay un campo finito subyacente asociado, en nuestro caso Z_q y Z_q^2.

La pregunta que debemos hacernos a continuación es: ¿ganamos algo definiendo el Problema del Logaritmo Discreto sobre curvas elípticas con respecto a la definición directa sobre los campos finitos subyacentes? La respuesta es que sí. Los matemáticos creen que la resolución del ECDLP es al menos tan compleja con la resolución del DLP. Pero además a este hecho hay que añadir una ventaja considerable: para mantener el mismo nivel de dificultad en la resolución del problema el valor de q que debemos elegir es mucho más pequeño en el caso de trabajar con curvas elípticas, lo que resulta muy interesante para

nuestro proyecto, ya que, como se ha comentado en repetidas ocasiones los endpoints objeto de estudio están equipados con procesadores de bajas prestaciones.

Para que efectivamente podamos obtener esta ventaja de la reducción de q, es necesario que elijamos el punto P correctamente. Debemos elegir un P tal que el orden r de ese punto en la curva sea lo más grande posible, asumiendo que nunca podrá ser mayor que el orden de la curva n. Una mala elección del punto P podría suponer que el orden de ese punto en la curva fuera un valor muy pequeño en comparación con q, es decir, que las posibles salidas del producto xP formaran un subgrupo con un número de puntos demasiado pequeño. Si por ejemplo estuviéramos trabajando con q=149 y eligiéramos un punto P tal que 7P = O, estaríamos hablado de únicamente siete posibles salidas como resultado de xP frente a los 149 elementos del campo finito Z_{149}.

Lógicamente para asegurar máxima dificultad en el Problema del Logaritmo Discreto en Curvas Elípticas seleccionaremos P de tal modo que r se aproxime lo máximo posible a q.

Aunque resulte obvio, para que el ECDLP tenga solución es necesario que P y Q pertenezcan al mismo subgrupo o, expresado en otros términos, P y Q deben ser linealmente dependientes.

Simplemente como referencia para el lector interesado, es comúnmente aceptado que la resolución del Problema del Logaritmo Discreto en una curva en la que r y q tengan un tamaño de 160 bits es al menos igual de complicado que resolver el mismo problema en Z_q donde q tiene 1024 bits.

Por último existe otra ventaja importante en ECDLP con respecto al DLP: el algoritmo *index-calculus* no funciona cuando se trabaja con curvas elípticas.

3.2.6. PROTOCOLO DIFFIE-HELLMAN SOBRE CURVAS ELÍPTICAS

Algunas páginas atrás estudiábamos el protocolo Diffie-Hellman y cómo este podía ser utilizado para establecer una clave entre dos extremos a través de un canal de comunicaciones público y de manera que solamente los extremos interesados la conozcan. Este mismo problema lo podemos definir cuando trabajamos con curvas elípticas y en este caso nos referiremos a él como ECDH (Elliptic Curve Diffie-Hellman).

Veamos nuevamente cómo funciona cuando José y Julián desean establecer una clave de sesión utilizando un canal que puede ser observado por cualquiera.

En primer lugar, y dado que estamos trabajando con curvas elípticas, José y Julián deben estar previamente de acuerdo en el uso de una curva concreta, es decir, deben compartir el conocimiento de los coeficientes a y b, un punto generador G (de orden r) y el valor q del campo finito subyacente Z_q.

Conocidos dichos parámetros veamos cómo tiene lugar el establecimiento de la clave entre ambos:

- En primer lugar José y Julián obtienen sendas parejas de clave privada y pública como sigue: Julián elige un número aleatorio k_{JU} comprendido entre 1 y r-1 y lo multiplica por G, obteniendo $P_{JU}= k_{JU}G$. La dupla (k_{JU},P_{JU}) son la clave privada y la clave pública de Julián. Del mismo modo José obtiene sus claves privada y pública (k_{JO},P_{JO}). Nótese que la clave privada es un número entero y la clave pública un punto de la curva elíptica.

- José y Julián ponen en conocimiento del otro (y de cualquiera) sus claves públicas P_{JO} y P_{JU}.

- Cuando José quiere iniciar una conversación con Julián, calcula un nuevo punto Q de coordenadas q_x y q_y multiplicando su clave privada (k_{JO}) por la clave pública de Julián (P_{JU}):

  $Q = k_{JO}P_{JU.}$

  Por su parte Julián calcula el mismo punto Q multiplicando su clave privada (k_{JU}) por la clave pública de José (P_{JO}).

- La clave secreta acordada será la coordenada q_x del punto Q.

- Q es el mismo punto, dado que $k_{JO}P_{JU} = k_{JO}k_{JU}G = k_{JU}k_{JO}G = k_{JU}P_{JO}$.

Puesto que la única información que José y Julián exponen son sus respectivas claves públicas, ninguna otra persona puede calcular la clave, salvo que sea capaz de resolver el problema del logaritmo discreto en curvas elípticas (ECDLP). Esta clave puede ser utilizada directamente como entrada para un algoritmo de cifrado simétrico o puede ser utilizada para derivar otra clave haciendo uso por ejemplo de un algoritmo de hash, que será la clave utilizada en el algoritmo de cifrado.

Al igual que sucede con el protocolo Diffie-Hellman definido sobre campos finitos, la versión sobre curvas elípticas también es vulnerable a los ataques de *man-in-the-middle*. Por lo tanto no es un protocolo seguro y es necesario añadir mecanismos extra que garanticen la autenticidad de los extremos. Existe una variante de ECDH conocida como ECDHE. La última E del acrónimo viene del término en inglés *ephimeral* y hace referencia a la volatilidad de las claves, que serán distintas en cada sesión. Por el contrario cuando se habla de ECDH se asume que las claves son estáticas.

3.2.6.1.ECDSA: FIRMAS EN CURVAS ELÍPICAS.

Aunque ya nos hemos referido a ellas anteriormente cuando repasábamos los algoritmos más conocidos de clave privada y de clave pública, dedicaremos ahora unas pocas líneas para definir las firmas digitales y para qué se usan.

En la vida real, la validación de identificación y autenticidad de escritos y documentos legales se determina por la presencia o ausencia de una firma manuscrita. En los sistemas informáticos, donde no existe el papel ni la posibilidad de incrustar una firma manuscrita, se hace necesario un mecanismo equivalente. Las firmas digitales son la respuesta a este problema.

El objetivo de dichas firmas es múltiple. Por un lado se persigue que el extremo receptor pueda verificar la identidad proclamada por el trasmisor. Por otro se debe garantizar el no repudio de los mensajes, es decir, que el trasmisor que confecciona un mensaje no pueda negar *a posteriori* que fue él quien construyó. Por último, las firmas también sirven para garantizar que el propio receptor no ha podido confeccionar el mensaje.

Al igual que la criptografía, las firmas digitales también se puede agrupar en firmas de clave secreta y firmas de clave pública. En el primer tipo, todos los usuarios confían en una entidad central que está en posesión de las claves de todos los usuarios y que por tanto tiene la capacidad de leer todos los mensajes.

El hecho de la existencia de semejante autoridad puede generar cierta desconfianza. Las firmas de clave pública pretenden resolver ese problema eliminando la necesidad de una autoridad central de confianza.

Recordemos que los algoritmos de cifrado de clave pública deben cumplir D(E(P))=P. Supongamos que además cumplen que E(D(P))=P, cosa que por ejemplo sucede en RSA. Si este es el caso, el usuario A puede enviar el texto normal firmado P al usuario B transmitiendo $E_D(D_A(P))$. El usuario A conoce su propia clave de descifrado (privada) y la clave de cifrado (pública) del usuario B. Cuando el usuario B recibe el mensaje utiliza su clave de descifrado para recuperar $D_A(P)$. Por último utilizando la clave pública de A obtiene P. En la siguiente imagen se ilustra la transmisión y recepción del mensaje entre los usuarios A y B.

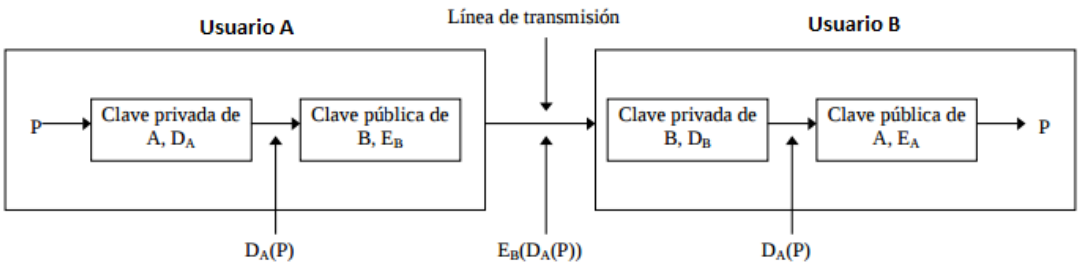


Ilustración 15 - Transmisión de mensaje firmado por A a B

Veamos ahora cómo funciona el mecanismo de firma. Supongamos ahora que el usuario A niega haber enviado el mensaje P al usuario B. Cuando el caso llega al tribunal de arbitraje pertinente, el usuario B puede presentar el mensaje P y también $D_A(P)$. Dicho tribunal puede comprobar si el mensaje P fue realmente confeccionado por A aplicando la clave pública de A. Si al aplicar dicha clave se obtiene el mismo mensaje P que el usuario B aseguró haber recibido de A, es incuestionable que dicho mensaje fue construido por A.

El sistema funciona siempre que A mantenga su clave privada en secreto y no la haya cambiado por otra, cosa que podría haber sucedido y ser perfectamente legal. Parece que de nuevo se llega a la necesidad de una entidad autorizada que registre los cambios de clave y la fecha en la que tienen lugar.

El algoritmo RSA se considera el estándar de facto y una gran cantidad de aplicaciones lo utilizan.

El algoritmo DSA es otro ejemplo de algoritmo de firma digital. Este algoritmo, al igual que RSA, opera con número enteros que pertenecen a campos finitos.

Cuando estamos trabajando con curvas elípticas, el algoritmo más comúnmente utilizado para producir y verificar firmas digitales es el conocido como ECDSA (Elliptic Curve Digital Signature Algorithm) y es en cierto modo el equivalente a DSA.

Veamos cómo funciona este algoritmo tanto para producir la firma como para verificarla.

Producción de la firma

Suponemos que los parámetros del sistema (la curva elíptica E y un punto generador P de orden r) ya está establecidos. Ahora José desea producir una firma para el mensaje que desea enviar a Julián. Para

ello elige un número aleatorio d_{JO} entre 1 y r-1, que será su clave privada. Seguidamente calcula su clave pública $Q_{JO} = d_{JO}P$.

Con todos estos datos ya puede firmar el mensaje m del siguiente modo:

- José elige un número aleatorio k entre 1 y r-1.
- Calcula un punto X de la curva como kP. Las coordenadas de este punto son (x_1,y_1).
- Calcula el valor $t = x_1 \bmod r$. Si t resulta ser igual a 0 se debe iniciar de nuevo el proceso eligiendo otro número aleatorio k distinto.
- Calcula $s=k^{-1}(HASH(m) + td_{JO}) \bmod r$. Si s resulta ser igual 0 se debe iniciar de nuevo el proceso eligiendo otro número aleatorio k distinto. HASH es una función de un solo sentido que mapea el mensaje m en una secuencia de bits de longitud fija. Típicamente se utiliza SHA-1, SHA-2 o SHA256. Si el algoritmo de hash elegido produce un resultado de longitud mayor que r bits será necesario truncar el resultado.
- La firma del mensaje es la dupla (t,s). José enviará a Julián el mensaje m (seguramente cifrado) junto con la firma ECDSA que acaba de calcular.

Verificación de la firma:

Julián recibe el mensaje de José junto con la firma (t,s). Julián conoce además todos los parámetros del sistema, la clave pública de José (el punto Q_{JO}) y el algoritmo HASH que se ha empleado para la generación de la firma.

Julián sigue los siguientes pasos para verificar la firma enviada por José y de este modo poder estar seguro de que fue realmente José y no un impostor quien envió el mensaje:

- Primero verifica que los valores t y s (que conforman la firma) están comprendidos entre 1 y r-1.
- Calcula h=HASH(m). Si m estaba cifrado, previamente lo habrá tenido que descifrar.
- Calcula los siguientes valores: $w = s^{-1} \bmod r$; $u_1 = hw \bmod r$; $u_2 = tw \bmod r$.
- Calcula el punto X de coordenadas (x_1,y_1) como $X = u_1P + u_2Q_{JO}$.
- Por último Julián calcula $v = x_1 \bmod r$.
- La firma es verificada solamente si t = v. En cualquier otro caso la firma no es válida y Julián no puede tener la certeza de que el mensaje m provenga de José.

3.2.6.2. SELECCIÓN DE UNA CURVA ELÍPTICA

Hasta aquí hemos visto qué son las curvas elípticas y cómo definir problemas complejos sobre ellas, en concreto el ECDLP, que son la base para construir sistemas criptográficos robustos. Pero ¿qué curva elíptica elegir? Dependerá de la aplicación y el tipo de cripto-sistema que se pretende construir. Hagamos un breve repaso de los parámetros o valores a elegir.

En primer lugar tenemos los coeficientes a y b de la ecuación ($E: y^2 = x^3 + ax + b$).

Por otra parte, tenemos la selección del campo subyacente (Z_q o Z_q^2), lo que se traduce en la elección de un determinado valor primo q grande.

En tercer lugar debemos seleccionar un punto P de la curva, que dará lugar a un subgrupo cíclico de orden r. Nos interesa un punto tal que el valor de r sea lo más cercano posible a q.

En el apéndice D de su publicación FIPS PUB 186-3, el NIST propone una serie de curvas definidas sobre Z_q para su utilización en sistemas criptográficos. En todas ellas el valor de a = -3 por razones de eficiencia de computo. Estas curvas son las siguientes:

Curva P-192:

q=6277101735386680763835789423207666416083908700390324961279 → 192 bits
r=6277101735386680763835789423176059013767194773182842284081 → 192 bits
b=2455155546008943817740293915197451784769108058161191238065 → 191 bits
Gx=602046282375688656758213480587526111916698976636884684818 → 189 bits
Gy=174050332293622031404857552280219410364023488927386650641 → 187 bits

Curva P-224:

q=26959946667150639794667015087019630673557916260026308143510066298881 → 224 bits
r=26959946667150639794667015087019625940457807714424391721682722368061 → 224 bits
b=18958286285566608000408668544493926415504680968679321075787234672564 → 224 bits
Gx=19277929113566293071110308034699488026831934219452440156649784352033 → 224 bits
Gy=19926808758034470970197974370888749184205991990603949537637343198772 → 224 bits

Curva P-256:

q=115792089210356248762697446949407573530086143415290314195533631308867097853951
→ 256 bits
r=115792089210356248762697446949407573529996955224135760342422259061068512044369
→ 256 bits
b=41058363725152142129326129780047268409114441015993725554835256314039467401291
→ 255 bits
Gx=48439561293906451759052585252797914202762949526041747995844080717082404635286
→ 255 bits
Gy=36134250956749795798585127919587881956611106672985015071877198253568414405109
→ 255 bits

Curva P-384:

p=39402006196394479212279040100143613805079739270465446667948293404245721771496870329047266088258938001861606973112319 → 384 bits
r=39402006196394479212279040100143613805079739270465446667946905279627659399113263569398956308152294913554433653942643 → 384 bits
b=27580193559959705877849011840389048093056905856361568521428707301988689241309860865136260764883745107765439761230575 → 384 bits
Gx=26247035095799689268623156744566981891852923491109213387815615900925518854738050089022388053975719786650872476732087 → 384 bits
Gy=8325710961489029985546751289520108179287853048861315594709205902480503199884419224438643760392947333078086511627871 → 382 bits

Curva P-521:

q=6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151 → 521 bits

r=6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449 → 521 bits

b=1093849038073734274511112390766805569936207598951683748994586394495931161507350160137087375737596232485921322967063133094384525315910129121423274884789859840 → 519 bits

Gx=2661740802050217063228768716723360960729859168756973147706671368418802944996427808491545080627771902352094241225065558662157113545570916814161637315895999846 → 520 bits

Gy=3757180025770020463545507224491183603594455134769762486694567779615544477440556316691234405012945539562144444537289428522585666729196580810124344277578376784 → 521 bits

Las cinco curvas anteriores P-NNN se diferencian esencialmente en el tamaño de q. Cada una de ellas recibe el nombre precisamente por el número de bits de q. Como es obvio, a mayor número de bits de q mayor seguridad, pero a su vez mayor tiempo de cómputo en las operaciones. Por tanto, se elegirá la curva adecuada buscando un equilibrio entre rendimiento y nivel de seguridad.

Los valores Gx y Gy corresponden con las coordenadas del punto generador que da lugar al subgrupo de orden r en la curva.

En el APÉNDICE C. FORTALEZA Y VALIDEZ TEMPORAL DE LAS CLAVES, se expone la equivalencia (en lo que a nivel de fortaleza se refiere) que proporcionan distintas longitudes de clave cuando se trabaja con algoritmos definidos directamente sobre campo finitos módulo un número primo con respecto a cuando se trabaja con curvas elípticas (que como ya sabemos operan sobre un campo finito módulo un número primo).

3.2.7. DEBILIDADES DE LAS CURVAS ELÍPTICAS

Hasta el momento hemos visto cómo funcionan las curvas elípticas y, aparentemente, todo son ventajas con respecto a los campos finitos sobre las que se definen. De hecho, la criptografía basada en curvas elípticas evolucionó considerablemente a comienzo de los noventa y su reputación no dejaba de mejorar, hasta que de forma repentina aparecieron unos algoritmos llamados *emparejamientos bilineales* que supusieron un frenazo en toda regla para esta tecnología.

Los emparejamientos bilineales, que trataremos en el próximo apartado, proporcionan un medio para reducir un ECDLP a un DLP. De este modo, resolviendo el DLP se resuelve el ECDLP. Y para resolver el DLP se puede emplear el algoritmo *index-calculus*. En resumen, la aparición de estos emparejamientos bilineales reduce de nuevo la seguridad conseguida con el uso de la tecnología de curvas elípticas a la proporcionada por el campo finito subyacente.

3.3. CRIPTOGRAFÍA BASADA EN IDENTIDAD

La Criptografía Basada en Identidad, también conocida por sus siglas del inglés IBC (Identitiy Based Criptography), es un tipo de criptografía de clave pública en la que los usuarios, en lugar de generar (de manera aleatoria) y utilizar un par de claves pública y privada, utilizan información relativa a ellos mismos. Si por ejemplo estos usuarios son personas, estaríamos hablando de utilizar información como su nombre,

apellidos, documento de identidad, o incluso información biométrica. Si los usuarios son PC esta información podría ser el número de serie de la placa base o del disco duro. En cualquier caso, se trata de utilizar cierta información que esté vinculada de una forma más o menos natural a una determinada entidad.

Muy al contrario de lo que se piensa, el concepto de criptografía basada en identidad surge hace muchos años. En concreto, es en 1984 cuando Adi Shamir propuso este tipo de sistema, aunque no pudo implementarlo ya que por entonces no existían los algoritmos ni las primitivas necesarias. En realidad Shamir sí que pudo definir las Firmas Basadas en Identidad, lo que permitía a los usuarios validar firmas digitales usando únicamente información pública de los mismos. Sin embargo el problema del Cifrado Basado en Identidad (también conocido por Identity Based Encryption o directamente por las siglas IBE) permaneció abierto durante muchos años. No fue hasta el año 2001 cuando Dan Boneh y Matthwe K. Franklin por un lado, y Clifford Cocks por otro, resolvieron el problema y crearon los primeros sistemas de Cifrado Basado en Identidad.

Boneh y Franklin basaron su solución en el conocido emparejamiento bilineal Weil Pairing sobre curvas elípticas, mientras que Cocks fundamentó la suya en el problema de los residuos cuadráticos.

La idea esencial de la criptografía basada en identidad es que los usuarios del sistema utilizan como clave pública determinada información vinculada a ellos de forma natural. La clave privada es construida por un Centro de Generación (de claves privadas), que asegura, después de establecer las comprobaciones que considere necesarias, que dicha clave privada ha sido generada para el usuario correcto.

Una de las ventajas principales de este tipo de sistemas es que prescinde del uso de certificados digitales como claves públicas, que es uno de los objetivos que perseguimos en este proyecto.

Utilizando este esquema de cifrado basado en identidad, José puede cifrar cierta información para Julián conociendo solamente el identificador público de Julián (supongamos que es su dirección de correo electrónico *julian@mimail.com*). Podría darse el caso de que Julián ni siquiera estuviera registrado en el sistema, es decir, ni siquiera tuviera su propia clave privada en el momento de recibir la información. En caso de querer descifrar la información, Julián debería presentarse ante el Centro Generador y obtener su propia clave privada una vez superadas las validaciones y comprobaciones que el Centro Generador tuviera a bien.

3.3.1. ESQUEMA DE BONEH-FRANKLIN

Hemos comentado unos párrafos más arriba que el sistema propuesto por Boneh y Franklin está basado en curvas elípticas y emparejamientos bilineales. El sistema criptográfico implementado para securizar las comunicaciones pre-boot de los endpoints bajo estudio está a su vez basado en este esquema, aunque se le han hecho algunas modificaciones para poder cumplir con los objetivos perseguidos. Antes de entrar a analizar dichas modificaciones, veamos cómo funciona el sistema propuesto por Boneh y Franklin.

El problema que hay que resolver es nuevamente el de enviar un mensaje cifrado desde José hacia Julián. No entraremos a ver cómo funciona el emparejamiento de Weil pues más adelante veremos que este va a ser sustituido por otro emparejamiento bilineal. Lo único que nos interesa por el momento es que el sistema está basado en un emparejamiento bilineal y, por lo tanto, vamos a trabajar con puntos que pertenecen a subgrupos distintos de la curva elíptica elegida como base del sistema. Así pues, trabajaremos con los grupos G1 y G2. Para definir el grupo G1 necesitaremos un punto generador P. Sin embargo, el

grupo G2 quedará definido por el propio emparejamiento, por lo que no necesitamos un punto generador para G2.

Boneh y Franklin eligieron la siguiente curva para la construcción del sistema:

E: $y^2 = x^3 + x$

El campo subyacente es Z_q, donde q es un número primo congruente con 2 módulo 3.

Los grupos G1 y G2 son de orden r.

El sistema propuesto establece un protocolo en cuatro fases: Establecimiento (Setup), Extracción (Extraction), Cifrado (Encryption) y Descifrado (Decryption).

En la fase de Establecimiento, el Centro de Generación de Claves, renombrado por estos autores como Private Key Generator (PKG) fija los parámetros que usará el sistema, tales como el punto generador del grupo G1, el valor del primo q del campo finito Z_q subyacente o la clave maestra del sistema, que debe permanecer totalmente en secreto.

La fase de Extracción se refiere a la obtención de las claves privadas para los usuarios a partir de su identificador público.

Las fases de Cifrado y Descifrado, como su propio nombre indican, son aquellas en las que se produce la transformación de texto normal en texto cifrado y viceversa.

Veamos un poco más en detalle cómo suceden estas cuatro fases al tiempo que analizamos el ejemplo en el que José y Julián intercambian cierta información de forma confidencial gracias al esquema de Boneh-Franklin.

En primer lugar, tiene lugar la fase de Establecimiento. En esta fase se establecen los siguientes valores en el Centro Generador:

- El punto generador P. Da lugar a la definición del grupo G1 y, en consecuencia, también G2.

- El valor del primo q para el campo finito subyacente Z_q.

- Un número aleatorio secreto s, que recibe el nombre de clave maestra del sistema y que nunca debe hacerse público. Este número podría guardarse de forma segura en un hardware criptográfico especializado del tipo HSM.

- El emparejamiento bilineal a utilizar. Recordemos que se trata de un algoritmo que opera sobre dos puntos de la curva y produce un resultado en el grupo G2.

- La clave pública del sistema asociada a la clave maestra. Esta se calcula como sP.

- Una función hash H1 para mapear una identidad pública con un punto en la curva que pertenece al grupo G1. Esta función es un hash en el sentido de que mapea una identidad pública en un punto de la curva sin que puedan ocurrir colisiones para identidades distintas.

- Una función hash H2 estándar, esto es, un algoritmo tal como MD5, SHA-1, SHA256, etc.

Una vez establecidos y dados a conocer todos estos parámetros (salvo la clave maestra s) los usuarios pueden empezar a hacer uso del sistema. Veamos como José y Julián son capaces de intercambiar información utilizando el sistema.

José, que es quien desea enviar el texto normal m a Julián, ejecuta la fase de extracción para su identificador, de modo que obtiene su clave privada:

- José proporciona su identificador ("jose@miemail.com") al Centro Generador de Claves. Este utiliza la primera de las funciones hash, H1, para convertir ese identificador en un punto de la curva: Jo = H1("jose@miemail.com").

- El centro generador de claves computa la siguiente clave privada para José: sJo, que es la multiplicación del secreto maestro por el punto al que la identidad pública de José ha sido mapeada.

- José recibe su clave privada sJo.

Por su parte, Julián repite el mismo procedimiento para obtener la clave privada sJu a partir de su identificador pública ("julian@miemail.com"). No es estrictamente necesario que Julián obtenga su clave privada aún. Podría hacerlo más adelante, por ejemplo después de recibir el mensaje cifrado de José.

A continuación José se dispone a cifrar el texto normal m, para lo cual procede del siguiente modo:

- Utiliza el identificador público de Julián (que no su clave pública) para mapear dicho identificador con un punto de la curva del grupo G1: Ju = H1("julian@miemail.com")

- José elige un número aleatorio r que pertenece a Z_q.

- Computa el emparejamiento bilineal para los puntos rJu y sP, que da como resultado el valor g que pertenece al subgrupo G2: g = e(rJu,sP).

- José utiliza ahora la segunda función hash, H2, para calcular el siguiente valor: m XOR H2(g), que es el texto cifrado c. XOR es la función lógica OR exclusiva.

- Por último José envía la dupla (rP, m XOR H2(g)) o lo que es lo mismo (rP, c)

Cuando Julián, ya en posesión de su clave privada sJu, recibe la dupla anterior, ejecuta la etapa de Descifrado como sigue:

- Julián calcula el valor como g = e(sJu, rP). Este valor g es el mismo que calculó José como e(rJu,sP). Este resultado es posible gracias a las propiedades de los emparejamientos bilineales. Veremos estas propiedades más adelante.

- Una vez que Julián está en posesión de este valor, calcula H2(g) XOR c, donde si sustituimos c por su valor obtenemos H2(g) XOR (m XOR H2(g)) que da como resultado el texto original m que José le envió a Julián.

Para que el sistema funcione hemos visto que se tiene que cumplir que e(rJu,sP) = e(sJu, rP) y esta magia sucede gracias a las propiedades de los emparejamientos bilineales.

3.3.2. EMPAREJAMIENTOS BILINEALES

En este apartado veremos en qué consisten los emparejamientos bilineales, sus propiedades y los diferentes tipos de emparejamientos bilineales que se conocen en la actualidad. No veremos cómo se computan ni las matemáticas que subyacen detrás de ellos, pues la materia sería suficiente como para realizar otro proyecto completo al margen. Confiaremos en que lo que los matemáticos y expertos en criptografía exponen es cierto y nos centraremos en aquello que nos interesa: cómo podemos hacer uso de los emparejamientos para construir sistemas criptográficos.

En primer lugar, comencemos por la definición. Un emparejamiento bilineal es un algoritmo o función que estable un mapeo entre dos puntos pertenecientes a dos grupos cíclicos en una curva elíptica y un elemento que pertenece a un tercer grupo, que es la extensión del campo finito Z_q, es decir, Z_q^k. Formalmente describiremos el emparejamiento así:

e: G1 x G2 → G3

Si G1 y G2 son el mismo grupo, es decir, si los puntos de la curva sobre los que opera el emparejamiento pertenecen al mismo subgrupo en la curva, diremos que el emparejamiento es simétrico. Si por el contrario G1 y G2 son grupos distintos, diremos que el emparejamiento es anti-simétrico.

Los emparejamientos bilineales cumplen las siguientes propiedades:

- **Bilinealidad**

  Sean P y Q puntos de la curva tales que P ϵ G1 y Q ϵ G2: $e(rP, sQ) = e(P, Q)^{rs}$

  Donde r y s son valores que pertenecen al campo finito subyacente Z_q o Z_q^2.

  Recordemos que rP y sQ son multiplicaciones escalares de un número por un punto y por lo tanto, cuando multiplicamos uno de los puntos por un número s, el resultado del emparejamiento se ve elevado a s.

- **No degeneración**

  Sean P y Q puntos de la curva tal que P ϵ G1 y Q ϵ G2: e(P,Q)≠1

  Estas propiedades son las que nos permiten construir sistemas basados en identidad.

Por otro lado, los emparejamientos bilineales solo se pueden aplicar para construir sistemas basados en identidad cuando trabajamos con curvas elípticas pero no en cualquier curva se pueden encontrar este tipo de emparejamiento.

Actualmente solo se conocen solamente dos emparejamientos bilineales sobre curvas elípticas: el emparejamiento de Weil y el emparejamiento de Tate, más conocidos por su denominación en inglés, Weil Paring y Tate Paring, respectivamente.

El primero de ellos es un emparejamiento lineal simétrico, mientras que el segundo es anti-simétrico. En general el emparejamiento de Tate es preferido por ser aproximadamente el doble de rápido y por presentar menos restricciones que el emparejamiento de Weil a la hora de ser utilizado en aplicaciones prácticas.

Una ventaja del emparejamiento de Weil es que es más sencillo de entender. Además produce una salida única. Por el contrario, el emparejamiento de Tate produce una salida que es en realidad una clase de equivalencia. Para aquellas aplicaciones en las que se necesite obtener un valor único, el problema se puede arreglar fácilmente mediante una operación de exponenciación. Aun asumiendo el coste computacional de esta exponenciación, el emparejamiento de Tate sigue siendo más rápido que el de Weil.

	Weil	Tate
Dependencia lineal	Simétrico	Anti-simétrico
Rendimiento	-	√
Aplicabilidad	-	√
Facilidad de comprensión	√	-

Tabla 2 - Comparación entre el emparejamiento de Weil y el emparejamiento de Tate

Tanto para el emparejamiento de Weil como para el de Tate existe una versión del algoritmo de Miller que permite su cálculo. Los detalles sobre cómo utilizar el algoritmo de Miller para ambos emparejamiento pueden obtenerse de la tesis de Martijn Mass, sobre criptografía basada en emparejamientos (ver aparatado de Referencias y Bibliografía).

En este proyecto utilizaremos el emparejamiento de Tate para la construcción del sistema criptográfico final.

Aunque la aparición de los emparejamientos bilineales supuso un importante impulso para la criptografía basada en identidad, inicialmente fueron vistos como un gran problema para los sistemas criptográficos basados en curvas elípticas. Finalizaremos nuestra discusión analizando el porqué.

3.3.2.1. EMPAREJAMIENTOS BILINEALES COMO ENEMIGOS DE LAS CURVAS ELÍPTICAS

En apartados anteriores enunciamos el Problema del Logaritmo Discreto (DLP) y su versión equivalente cuando se trabaja con curvas elípticas (ECDLP).

Enunciemos ahora otro de los denominados problemas de difícil solución: el Problema de Decisión de Diffie-Hellman, también conocido por sus siglas en inglés ECDDHP (Elliptic Curve Decisional Diffie-Hellman Problem). La formulación de dicho problema es como sigue:

Dados P, rP, sP y tP, decidir si rs = t.

Si para la curva con la que se está trabajando existe un emparejamiento bilineal, si e(P,tP) es igual a e(rP,sP), entonces rs = t.

De modo similar, con la presencia de dicho emparejamiento, el Problema del Logaritmo Discreto también resulta sencillo de resolver. Recordemos que en este problema tenemos los puntos P y Q y queremos obtener el valor x tal que xP = Q. Con la presencia del emparejamiento en la curva podemos obtener fácilmente el valor de x transformando el problema ECDLP en un problema DLP convencional. Para ello simplemente calcularemos los valores: $e(P,P)=g$ y $e(P,Q)=e(P,xP)=g^x$.

Y una vez reducido el problema podremos aplicar el algoritmo index-calculus para obtener el valor de x.

3.3.2.2. ELECCIÓN DE CURVAS ELÍPTICAS ADECUADAS

Los emparejamientos de Weil y Tate solamente pueden ser aplicados cuando el grado de inmersión de la curva es lo suficientemente pequeño (menor o igual que 6). Por este motivo, no vale con elegir los parámetros a y b que definen la curva elíptica al azar, puesto que el grado de inmersión suele resultar en un valor muy superior. Por este motivo habremos de elegir nuestra curva elíptica de un grupo especial que recibe el nombre de curvas supersingulares.

Veamos qué significa que una curva sea supersingular. Supongamos que tenemos una curva E definida sobre el campo finito Z_q. Recordemos que el orden de la curva, n, es el número de puntos de la curva. Se define la traza del endomorfismo de Frobenius como el valor t que hace que se cumpla la siguiente igualdad: $n = q + 1 – t$, siendo q el número de elementos del campo finito Z_q y $q=p^m$ donde p es un número primo. Si p divide t, entonces E es una curva supersingular. Nótese que en esta definición q no es un número primo directamente, sino que es una potencia de p, que sí es primo.

Por otra parte, las curvas supersingulares tienen otra importante ventaja: permiten mapear un punto P de orden r con otro punto Q, también de orden r, que es linealmente independiente. Esto resulta muy interesante a la hora de aplicar el emparejamiento de Tate, que, como ya se ha mencionado, es anti-simétrico y por tanto opera sobre puntos que son linealmente independientes.

3.4. RESUMEN Y CONCLUSIONES

En este capítulo hemos revisado los conceptos básicos de la criptografía moderna ilustrando las diferencias esenciales que existen entre los sistemas de clave pública y los de clave privada.

Hemos enunciado y analizado el Problema del Logaritmo Discreto, que, por el hecho de ser un problema matemático de difícil solución, constituye la base de muchos sistemas criptográficos, como por ejemplo el protocolo Diffie-Hellman, que tiene como objetivo el establecimiento de una clave común entre dos extremos que solamente pueden utilizar un canal de comunicación observable por terceros.

Después, nuestro anaísis nos ha llevado al estudio de la Criptografía de Curva Elíptica, descubriendo que esta tiene algunas ventajas con respecto a la criptografía que trabaja directamente sobre campos finitos de números enteros. La principal ventaja es que nos va a permitir emplear números más pequeños para proporcionar los mismos niveles de seguridad que proporcionan los sistemas que trabajan directamente sobre los campos finitos. Obviamente esta ventaja se traduce en que las operaciones matemáticas consumen menor tiempo y requieren en consecuencia procesadores de menores prestaciones, lo que resulta ideal para los endpoints.

Hemos visto cómo se traslada y se trabaja con el Problema del Logartimo Discreto cuando los sistemas se sustentan en Criptografía de Curva Elíptica. A continuación hemos presentado unos algoritmos ciertamente especiales, conocidos como Emparejamiento Bilineales, cuya aparición supuso inicialmente un grave problema para la Criptografía de Curva Elíptica, aunque poco después se descubrió que abrían camino a nuevos sistemas criptográficos y posibilitaban la resolución de problemas que llevaban años planteados y sin solución, como era el caso de la Criptografía Basada en Identidad. La Criptografía Basada en Identidad es un nuevo tipo de criptografía de clave pública en el que la clave pública de un determinado individuo o sistema se vincula a este de forma natural, y no mediante un certificado digital que una autoridad de certificación, en la que todo el mundo confía, genera para dicho individuo.

Dentro del estudio de este nuevo concepto de sistemas de clave pública, ilustramos el sistema de cifrado basado en identidad diseñado por los autores Boneh y Franklin.

Al final del capítulo definimos de forma más precisa el concepto de emparejamiento bilineal e ilustramos los dos emparejamientos conocidos hasta la fecha: el de Weil y el de Tate.

Finalmente acabamos ofreciendo las directrices básicas para la selección de curvas elípticas adecuadas para la construcción de sistemas criptográficos.

Este capítulo constituye la base teórica necesaria para diseñar la solución cuyo diseño y desarrollo se describe en los capítulos 4 y 5.

4. DISEÑO DE LA SOLUCIÓN

Aunque hasta este punto se ha ido esbozando el sistema diseñado es el momento de poner todas las piezas juntas para dar lugar a la solución de comunicaciones seguras en tiempo de pre-boot.

No perdamos de vista que lo que pretendemos es dotar a los endpoints objeto de estudio de la capacidad de comunicarse con otros equipos de forma segura. En general, y para hacer la lectura más fácil de entender, estaremos hablando de tres equipos o sistemas. El primero de ellos es el propio endpoint, al que nos referiremos también como cliente. El segundo es el extremo con el que endpoint pretende establecer la comunicación. Podría tratarse de otro endpoint, pero en general será algún servidor, bien sea de imágenes binarias, de dominio, de DHCP, etc. Por este motivo nos referiremos a él simplemente como servidor. Por último, tendremos un Centro Generador de Claves, que es el que establece los parámetros del cripto-sistema para que estas comunicaciones seguras sean posibles entre el cliente y el servidor.

El diseño que se va a exponer a continuación ha sido implementado sobre el kernel de iPXE, dando lugar a un prototipo básico completamente funcional.

Una vez vistos los tres componentes sobre los que la solución aplica (cliente, servidor y Centro Generador) es momento de exponer las etapas o funciones básicas sobre las que se sustenta la solución. El sistema criptográfico va a contar con tres etapas o funcione bien diferenciadas:

- Establecimiento.

- Obtención de clave privada.

- Conexión (autenticación mutua con acuerdo de clave de sesión).

En la fase de establecimiento es donde se fijan los parámetros del cripto-sistema, esto es, la curva elíptica a utilizar, el punto generador, el secreto maestro, etc.

En la fase de obtención de clave se define la obtención de las claves privadas para los extremos intervinientes en la comunicación (cliente y servidor). Cada uno de los extremos deberá acudir al Centro Generador para obtener su clave privada.

Por último en la fase de conexión es donde se produce la autenticación de los extremos y el establecimiento de una clave de sesión que será la utilizada para cifrar la comunicación.

Veamos por separado cómo funciona cada una de ellas.

4.1. ESTABLECIMIENTO DEL CRIPTO-SISTEMA

El establecimiento del cripto-sistema define en cierto modo las reglas del juego. El Centro Generador decide la curva elíptica a utilizar, el campo finito subyacente, el punto generador, el secreto maestro y los algoritmos de hash H1 y H2 presentados en la sección 3.3.1. Adicionalmente se requiere el uso de una tercera función hash HashNP que describiremos en el presente apartado. Al conjunto de todos estos datos los denominaremos *parámetros del cripto-sistema*.

En el prototipo construido se han seleccionado los siguientes parámetros:

- **Curva elíptica E**
  Se ha seleccionado la curva supersingular P-256 recomendada por el NIST en FIPS 186-3 siendo b=0. Esta curva es E: $y^2 = x^3 - 3x$. Por lo que implícitamente a=-3.

- **Punto generador P** (de orden r expresado por sus coordenadas Gx y Gy en hexadecimal):
  P (6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0f4a13945 $d898c296_h$,
  4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 $37bf51f5_h$).

- **Orden r** (tamaño del subgrupo cíclico definido por el punto generador):
  1157920892103562487626974469494075735299969552241357603424222590610685120443 69.

- **Secreto maestro s** (número entero cualquiera entre 1 y r). Una vez elegido hay que mantenerlo fijo.

- **Clave pública del sistema**: sP.

- **Función hash H2**: SHA256

- **Función hash H1 (*mapIdentity*)**: esta función (de un solo sentido) se utiliza para mapear una identidad real en un punto de la curva elíptica E. Por eso nos referiremos a ella también como *mapIdentity.* La función calcula el punto en la curva como sigue. En primer lugar, convierte la identidad real (cadena de texto) en un número utilizando el algoritmo SHA256 (podría haberse utilizado cualquier otro algoritmo de hash sin colisiones). Ese número se considera la coordenada x del punto en la curva con el que estamos estableciendo la relación. A partir de "x" se calcula "y", utilizando la ecuación de la curva elíptica. La identidad real de los endpoints es el número de serie del disco duro, que directamente es un número (típicamente representado en hexadecimal), por lo que ni siquiera habría sido necesario aplicar el algoritmo SHA256 para obtener un número. En cualquier caso, pensando en usos futuros que podría tener el sistema en construcción, se aplicará el algoritmo SHA256 a la identidad real que se desea mapear.

- **Función hash HashNP:** esta función mapea un número n y un punto P en otro número h. Nuevamente se trata de una función que opera en un único sentido. La función HashNP toma los bits de n y los concatena con los bits de las coordenadas del punto P. El resultado de la concatenación de todos esos bits se utiliza como entrada al algoritmo SHA256, produciendo un número de longitud 256 bits. Ese número módulo r es la salida de la función HashNP.

Todos estos parámetros (a excepción del secreto maestro s) son públicos y han estar en conocimiento de todos los equipos que pretendan hacer uso del cripto-sistema de comunicaciones seguras. En el equipo cliente han sido incluidos dentro del propio kernel de iPXE como datos estáticos. En el equipo servidor también han sido incluidos como datos estáticos, aunque la mayor parte de ellos pueden almacenarse simplemente en un fichero de configuración.

4.2. OBTENCIÓN DE LA CLAVE PRIVADA

Cualquier equipo que pretenda hacer uso del cripto-sistema construido para mantener comunicaciones seguras con otros equipos deberá estar en posesión de su clave privada. Para ello, deberá contactar con el Centro Generador de Claves o PKG, identificarse ante él con su identidad real (pública) y recuperar su clave privada a partir de esta. Obviamente, esta comunicación entre el equipo que solicita su clave privada y el PKG no puede ser asegurada por el cripto-sistema, luego es necesario recurrir a otros esquemas y mecanismos para poder asegurar esta comunicación.

En primer lugar, en el cripto-sistema construido el Centro Generador cuenta con una lista blanca de aquellos equipos que pueden presentar una solicitud de clave privada. Esta lista blanca está conformada por los identificadores públicos de los equipos. Es muy importante que dicha lista blanca esté debidamente custodiada y protegida frente a modificaciones indebidas. Queda fuera del alcance de este proyecto la protección de la lista blanca de identidades reales. Para el caso concreto de los endpoints, la lista blanca estará formada por los números de serie de sus discos duros. De este modo, el Centro Generador cuenta con un primer elemento de filtrado o barrera de seguridad. En caso de recibir una petición de clave privada por parte de un equipo cuya identidad pública (número de serie del disco duro) no figure en dicha lista blanca, el Centro Generador rechazará dicha petición. Lógicamente, el primer tipo de ataque queda servido: si algún sistema consigue emular el identificador del disco duro de un disco legítimo, podría obtener la clave privada. Existen distintos mecanismos para combatir este tipo de ataques de suplantación de identidad, como por ejemplo el uso de una OTP o un PIN. No nos detendremos aquí a evaluarlos.

Para finalizar la etapa de obtención de la clave privada es necesario hablar de su almacenamiento. Si la clave privada no es almacenada, esto supone recuperarla cada vez que se quiera establecer una comunicación segura. Todo esto conlleva un coste computacional y resulta conveniente almacenar la clave privada, pero, dado que el kernel de iPXE no tiene la capacidad de guardar información en el sistema de ficheros del endpoint, nos encontramos ante una importante disyuntiva: o bien no almacenar la clave privada y obtenerla del Centro Generador cada vez que se quiera establecer una conexión segura, o bien proporcionar al kernel de iPXE funciones para el acceso al disco. A pesar del esfuerzo adicional que ha supuesto, hemos considerado conveniente optar por la segunda alternativa. El cómo se ha conseguido dotar al kernel de iPXE de las funciones de lectura y escritura en disco se expone en la sección 5.4 ESCRITURA EN DISCO EN ENSAMBLADOR de la presente memoria. La clave privada se almacena en uno de los denominados sectores ocultos del disco. Para mayor seguridad se almacenará cifrada con el algoritmo AES-256 utilizando como clave (simétrica) el número de serie del disco duro.

Presentemos finalmente el protocolo de obtención de clave privada. Recordemos que la comunicación va a tener lugar en un cliente (endpoint) y el Centro Generador de Claves. El siguiente esquema representa el intercambio mantenido entre las partes para la obtención de la clave privada del endpoint.

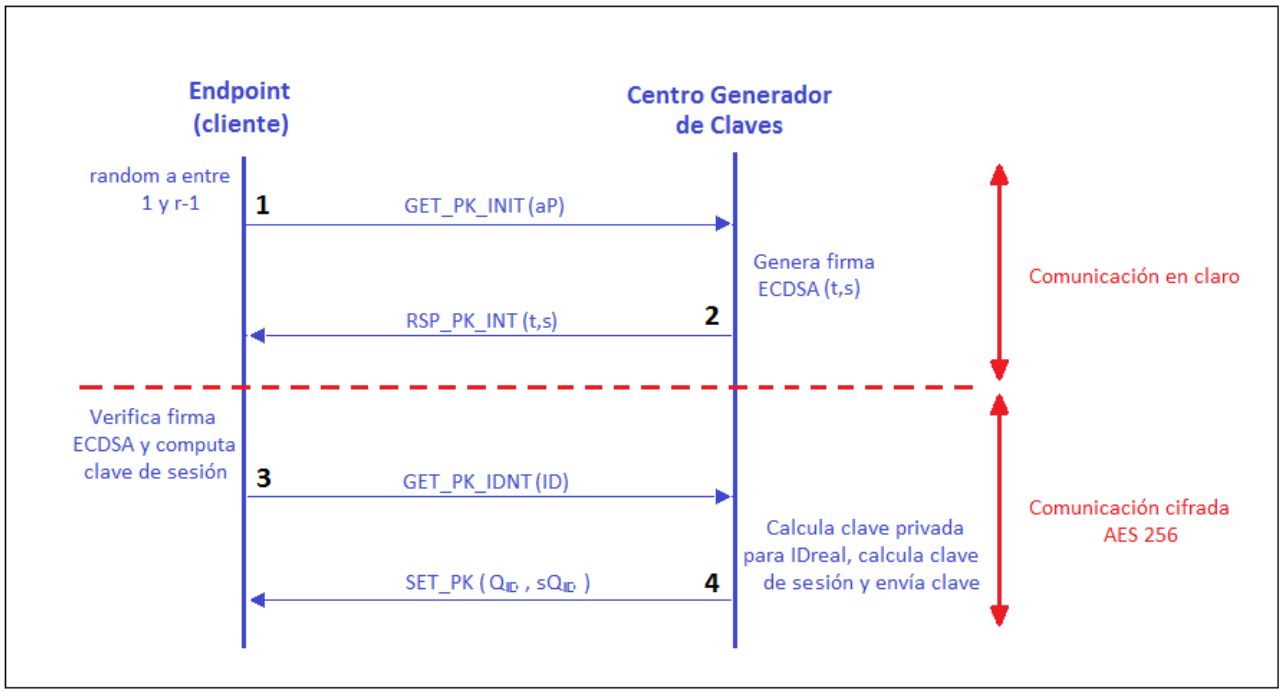


Ilustración 16 - Esquema de obtención de clave privada de cliente en el cripto-sistema

1. El extremo cliente (endpoint) genera un número aleatorio a (comprendido entre 1 y r-1) y envía el punto aP al Centro Generador a través de un canal abierto y potencialmente inseguro.

2. El Centro Generador recibe el punto aP y envía la firma ECDSA de la coordenada x de dicho punto. La firma ECDSA consiste en la dupla (t,s), como se vio en el apartado *ECDSA: FIRMAS EN CURVAS ELÍPICAS.* No confundir esta s de la firma ECDSA con la clave maestra del cripto-sistema. De hecho, el Centro Generador utiliza la clave maestra s como clave privada para producir la firma ECDSA. Recuérdese que en el proceso de generación de la firma el Centro Generador elige un número aleatorio k.

3. El cliente recibe y verifica la firma ECDSA utilizando sP como clave pública del Centro Generador. Si la firma resulta válida, el cliente ha autenticado al Centro Generador de Claves, pues solamente él pudo haber generador la firma ECDSA utilizando su clave privada s, que es el secreto maestro. Durante la verificación de la firma ECDSA el cliente obtiene el punto kP y a partir de él calcula el punto akP. Sobre la coordenada y de ese punto calcula el hash SHA256, obteniendo un número de 256 bits que se utilizará como clave para cifrar la comunicación restante utilizando AES-256 como algoritmo de cifrado simétrico. Finalmente, el cliente envía su identidad pública real (número de serie del disco duro en representación hexadecimal) a través del canal cifrado.

4. El Centro Generador recibe la información cifrada. Calcula el punto kaP del que deriva la clave de sesión del mismo modo que hizo el cliente. Descifra los datos obteniendo el identificador real del cliente, ID. A continuación chequea la lista blanca de identificadores de endpoint para comprobar que el número de serie del disco duro que presenta el cliente está registrado como legítimo. Seguidamente mapea el identificador real ID en el punto de la curva Q_{ID}. Diremos que el punto Q_{ID} es la identidad del cliente en el cripto-sistema basado en identidad. El Centro

Generador calcula ahora la clave privada para la identidad del cliente como sQ_{ID}. Finalmente, envía la identidad del cliente en el sistema (Q_{ID}) y la clave privada (sQ_{ID}) al endpoint a través del canal cifrado. Nótese que Q_{ID} podría haberlo computado directamente el cliente siempre que conozca el algoritmo para mapear una identidad real o física en un punto de la curva elíptica utilizada por el cripto-sistema, es decir, siempre que conozca la función hash *mapIdentity*.

4.3. COMUNICACIÓN AUTENTICADA Y CIFRADA

Una vez expuestas las fases de establecimiento del cripto-sistema y de obtención de claves por parte de los extremos interesados en mantener comunicaciones seguras, ya estamos en disposición de definir la última fase, en la que se produce realmente la autenticación de los extremos y el establecimiento de un canal cifrado para el intercambio de datos, todo ello gracias a la Criptografía Basada en Identidad expuesta en apartados anteriores.

En adelante, supondremos que es el extremo cliente el que inicia la conexión con el servidor, aunque a la inversa el sistema funcionaría exactamente del mismo modo.

Tanto el cliente como el servidor ya cuentan con sus claves privadas, que son respectivamente sQ_{ID} y sX_{ID}. Por tanto estamos asumiendo que la identidad real del cliente ha sigo mapeada en el punto Q_{ID} de la curva, y la identidad real del servidor ha sido mapeada en el punto X_{ID}.

El establecimiento de la conexión segura (cifrada y mutuamente autenticada) tiene lugar según el intercambio mostrado a continuación.

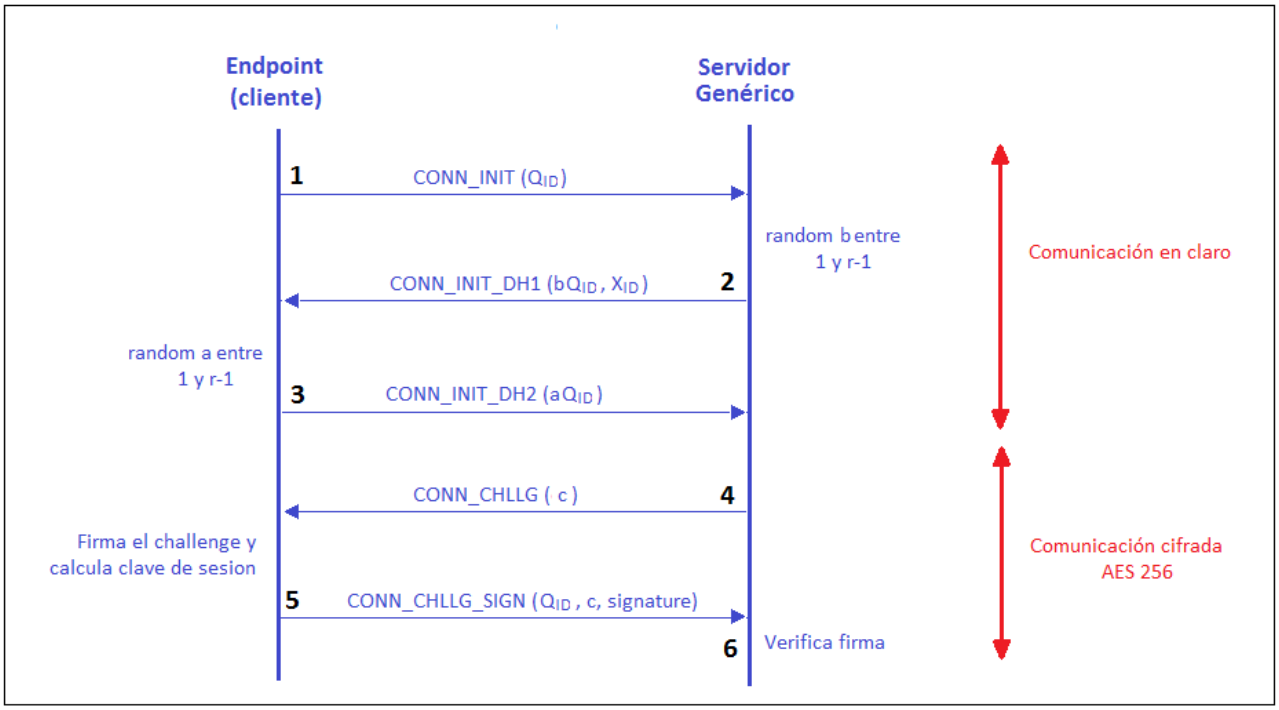


Ilustración 17 - Esquema de establecimiento de una comunicación segura (autenticada y cifrada)

1. El extremo cliente (endpoint) abre la conexión contra el extremo servidor y le envía su identificador público en el cripto-sistema, es decir, Q_{ID}.

2. El servidor genera un número aleatorio b comprendido entre 1 y r-1, calcula bQ_{ID} y envía el punto calculado al cliente. Además el servidor envía su identificador público en el sistema, esto es, el punto X_{ID}.

3. El cliente recibe el punto bQ_{ID}, calcula otro número aleatorio a comprendido entre 1 y r-1 y envía al servidor aQ_{ID}.

4. Cuando el servidor recibe $aQ_{ID,}$ genera un challenge c que envía al cliente. El challenge es simplemente un número aleatorio en Z_q que el servidor propone al cliente para que este lo firme. Al punto aQ_{ID} lo denominaremos M por simplicidad. Además, a partir de este punto la información se envía cifrada aplicando el algoritmo AES-256. La clave de sesión k_{sess}, utilizada como entrada para el algoritmo de cifrado simétrico, se calcula del siguiente modo:
   - $k = e(sX_{ID}, aQ_{ID})$
   - $k' = concat\ (k)$
   - $k_{sess} = SHA\text{-}256(k)$

   k' se obtiene como la concatenación de los bits de la parte real y la parte imaginaria de k. Recordemos que k es la salida del emparejamiento bilineal y por tanto se trata de un número complejo.

5. Cuando el cliente recibe los datos, lo primero que tiene que hace es descifrarlos. Para ello necesita recuperar la misma clave de sesión (clave AES) que utilizó el servidor para cifrarlos. Procede del siguiente modo:
   - $k = e(X_{ID}, sQ_{ID})^a$
   - $k' = concat\ (k)$
   - $k_{sess} = SHA\text{-}256(k)$

   Ahora el cliente extrae el challenge c descifrando los datos recibidos. Por el hecho de poder descifrar la información, el cliente ya está autenticando al servidor, pues únicamente este puede haber generado dicha clave de sesión (al estar en posesión de la clave privada sX_{ID}).
   Para firmar el challenge c, el cliente calcula $h = HashNP\ (c, aQ_{ID})$. Después calcula el punto $N = (a + h)sQ_{ID}$. La firma son los puntos M y N. Como M es un valor que el cliente ya ha enviado al servidor, en este paso el cliente podría decidir enviar solamente N. En cualquier caso, envíe solo N, o envíe M y N, la firma la constituyen ambos puntos. Junto con la firma, el cliente envía el propio challenge que fue firmado simplemente para fines de comprobación. Estos datos los vuelve a enviar cifrados utilizando de nuevo el algoritmo AES-256 y la misma clave de sesión que computó para descifrar el mensaje recibido del servidor.

6. Finalmente, cuando el servidor recibe los datos los descifra, obteniendo de este modo el challenge y la firma (puntos M y N). Para verificar la firma procede del siguiente modo:
   - Calcula el número $h = HashNP(c, aQ_{ID})$.
   - Verifica que $e(P,N) = e(sP, aQ_{ID} + hQ_{ID})$.

   Si la firma es verificada, el servidor puede estar seguro de que está hablando con el cliente.

A partir de este momento el cliente (endpoint) y el servidor se han autenticado mutuamente y han intercambiado una clave de sesión que les permite intercambiar datos de forma segura.

4.4. RESUMEN Y CONCLUSIONES

En este capítulo hemos diseñado la solución del sistema criptográfico que queremos desarrollar para dotar a los endpoints de comunicaciones seguras en tiempo de pre-boot sin certificados. Podríamos decir por tanto que es el capítulo más importante de toda la memoria.

Hemos llegado a una solución que se implementa en tres etapas o fases: Establecimiento, Obtención de Clave Privada y Conexión.

La fase de Establecimiento está destinada a fijar los parámetros del cripto-sistema para que todos los intervinientes estén en posesión de los datos y reglas necesarias para participar de dicho sistema. Estos parámetros son la curva elíptica, el punto generador, el secreto maestro (que únicamente conoce el Centro Generador), los algoritmos de hash a utilizar, etc.

En la fase de Obtención de Clave Privada se define el protocolo por el que todo aquel sistema que quiera participar del cripto-sistema basado en identidad obtiene su clave privada del Centro Generador de Claves. En este protocolo se hace uso del algoritmo de firma ECDSA para autenticar al Centro Generador.

La fase de Conexión es aquella en la que definitivamente los extremos de la comunicación establecen una conexión segura. En esta conexión los extremos se autentican mutuamente y acuerdan una clave de sesión común que será utilizada para cifrar los datos intercambiados mediante un algoritmo de cifrado simétrico, en concreto, AES-256.

Este diseño se implementa dentro del kernel de iPXE estudiado en el capítulo 2. Los detalles de cómo se lleva a cabo la codificación de este diseño se exponen en el capítulo 5.

5. DESARROLLO

Una vez definida la solución veremos cómo se ha desarrollado el prototipo que la implementa. No entraremos a analizar detalles de la implementación de los algoritmos o del código en sí ya que estos aspectos se rigen por los sistemas de calidad y desarrollo de la empresa en la que se ha desarrollado el prototipo, sino en cómo se ha desarrollado e integrado la solución, cuáles son las piezas que la componen y qué herramientas o recursos se han utilizado.

5.1. MEDIOS UTILIZADOS

Durante la fase de implementación del prototipo se han utilizado las siguientes herramientas:

- **Máquinas virtuales**
  Han sido imprescindibles para la realización de las pruebas durante el desarrollo y después de él. Haber utilizado equipos físicos para instalar y testar código MBR y de pre-boot habría sido sencillamente imposible. Cada error cometido habría supuesto la reinstalación del sistema operativo en disco. Lógicamente la instalación de un sistema operativo es una tarea que puede llevar muchos minutos, incluso horas. Las máquinas virtuales nos han permitido emular un equipo físico y por cada intento fallido solamente ha sido necesario restaurar la máquina a su estado anterior.
  El software de virtualización utilizado ha sido Virtual Box en su versión 4.3.

- **Entorno de Desarrollo y Compilador**
  El prototipo ha sido realizado íntegramente en los lenguajes *C* y en ensamblador en línea. Para el desarrollo se ha utilizado el editor avanzado *Emacs* con realzado de sintaxis específico para estos lenguajes de programación.
  El compilador utilizado ha sido *gcc (GNU C Compiler) v4.6*. Este compilador permite la compilación de código ensamblador en línea, lo que significa que dentro del lenguaje de más alto nivel (*C* en este caso) es posible embeber código ensamblador. La sintaxis del código ensamblador no es la de Intel, sino la de AT&T, dado que es la que entiende *gcc*.

- **Herramientas de depuración**
  Durante el desarrollo del prototipo se ha utilizado constantemente la herramienta de depuración *gdb.* Es importante resaltar que, al tratarse de un kernel de pre-boot lo que se depura, ha de utilizarse un cable serie para conectar el equipo en el que se encuentra el depurador *gdb* con el equipo remoto (endpoint). Como el endpoint utilizado durante el desarrollo es una máquina virtual, es necesario virtualizar igualmente la conexión mediante cable serie. Dada la relevancia de esta técnica combinada con el uso de la virtualización de equipos, dedicaremos el siguiente capítulo a ilustrar en detalle cómo se realiza dicha tarea.

- **Editor de discos**
  La aplicación *HxD Editor* permite abrir un disco duro para examinarlo byte por byte. El disco duro a analizar puede ser el propio disco desde el que se está ejecutando el programa HxD. Esta herramienta ha sido muy útil para analizar el código MBR que se escribe en el registro maestro de arranque y sucesivos.

- **Red de comunicaciones**
  La red de comunicaciones utilizada durante el desarrollo del prototipo ha sido una red TCP/IP local formada por un los siguientes elementos:
  - Un router con servidor DHCP integrado.
  - Endpoint y servidor virtualizados: máquinas virtuales con S.O. Windows XP y Windows 7.
  - Máquina en la que se ejecuta el Centro Generador de claves privadas.

- **Herramientas de análisis de tráfico**

  También conocidas como sniffers de red. Estas herramientas permiten interceptar y analizar el tráfico de red que intercambian los extremos sin modificarlo. Se han utilizado las herramientas tcpdump y Wireshark para inspeccionar los intercambios en los elementos intervinientes en las comunicaciones (endpoints, Centro Generador de Claves y servidor genérico).

- **Otras herramientas**

  También se han utilizado otras herramientas auxiliares para calcular valores de hashes o cifrados simétricos. Existen multitud de herramientas online que permiten realizar este tipo de operaciones sin necesidad de instalar aplicaciones en el equipo de desarrollo.

5.2. LIBRERÍA CRIPTOGRÁFICA C INTEGRADA EN iPXE

La solución diseñada en el capítulo 4 se ha implementado en una librería C compuesta por una serie de ficheros fuente (.c) y ficheros de cabecera (.h). Esta librería se ha codificado dentro del proyecto de iPXE, e intencionadamente se ha mantenido un vínculo débil con el resto de fuentes del proyecto iPXE original con el objetivo de poder desacoplarla y extraerla fácilmente para su uso en cualquier otro proyecto software codificado en el lenguaje de programación C. Este vínculo débil se reduce esencialmente a la utilización de los algoritmos estándar de cifrado o hash, tales como AES, SHA1 o SHA256, cuya implementación proporciona el proyecto original de iPXE. Se ha considerado conveniente utilizarlos y no reimplementarlos.

De hecho, para poder probar el sistema completo se ha desarrollado un programa que realiza la función de Centro Generador de Claves y otro programa que ejerce el rol de servidor genérico, cuya única función es la establecer una conexión segura con el endpoint/cliente (kernel de iPXE). Estos programas han sido realizados por otro equipos de personas dentro de la empresa apoyándose en la librería C. Puesto que la única dependencia que tiene la librería con el proyecto de iPXE es la implementación de los algoritmos de cifrado AES y de los algoritmos de hash, una implementación de estos algoritmos es lo único que se ha de incluir para el uso de la librería. Para concretar un poco más diremos que el Centro Generador de Claves y el servidor genérico han sido implementados en Java y se ha recurrido al uso de la tecnología de interfaz JNA para poder utilizar la librería C desarrollada en este proyecto.

A modo de resumen presentamos en la siguiente tabla un listado de los principales ficheros fuente que se han implementado, dónde quedan ubicados dentro de la estructura del proyecto iPXE y cuál es la función que desempeñan dentro de la solución.

FICHERO	RUTA EN PROYECTO iPXE	DESCRIPCIÓN
ibcbigdigits.c	./src/ibc/	Implementación de las funciones para manejar números grandes. Este código es parte de la librería "BigDigits multiple-precision arithmetic library Version 2.2", escrita y mantenida por David Ireland.
bigdtypes.c	./src/ibc/	Redefiniciones de tipos numéricos como uint16 o uint32.
curve.c	./src/ibc/	Implementación del tipo de dato Curva Elíptica y de las operaciones para trabajar con este tipo de dato.
hsm.c	./src/ibc/	Wrapper que emula un HSM para el almacenamiento y custodia de claves.
ibcsha1.c	./src/ibc/	Implementación alternativa a la proporcionada por el proyecto de iPXE para el algoritmo de hash SHA1.
ibehelper.c	./src/ibc/	Implementación de funciones criptográficas auxiliares.
line.c	./src/ibc/	Implementación del tipo de dato "recta" y de las operaciones para trabajar con este tipo de dato.
point.c	./src/ibc/	Implementación para la definición y operación con puntos de una curva elíptica.
complex.c	./src/ibc/	Implementación del tipo de dato "número complejo" y de las operaciones para trabajar con este tipo de dato.
bigdigitsRand.c	./src/ibc/	Generador de números aleatorios grandes.
ibc_interface.c	./src/ibc/	Funciones exportadas de forma pública para ser utilizadas por los programas y módulos que participen del cripto-sistema.
ibetypes.c	./src/ibc/	Tipos compuestos de datos definidos para ser utilizados por los programas y módulos que participen del cripto-sistema.
disk.c	./src/ibc/	Funciones desarrolladas en Ensamblador en Línea que proporcionan acceso básico de lectura y escritura a sectores del disco duro.
util.c	./src/ibc/	Funciones auxiliares de utilidad para todos los módulos que participan del cripto-sistema.
ibcpeer.c	/src/net/udp/	Implementación principal de las funciones del cripto-sistema: establecimiento o inicialización, obtención de clave privada y establecimiento de conexión segura.

Tabla 3 - Ficheros fuente de la solución

Los respectivos ficheros de cabecera contienen tanto la declaración de prototipos de las funciones como la de las estructuras de datos empleadas. Todos los ficheros de cabeceras se encuentran en la ruta ./src/includes/ibc del proyecto de iPXE.

En la siguiente tabla se listan los ficheros fuente originales del proyecto de iPXE que contienen algoritmos criptográficos y que han sido reutilizados en el desarrollo del prototipo.

FICHERO	RUTA EN PROYECTO DE iPXE	DESCRIPCIÓN
axtls_aes.c	./src/crypto	Implementación del algoritmo de cifrado AES-256-CBC
sha256.c	./src/crypto	Implementación del algoritmo de hash SHA256
sha1.c	./src/crypto	Implementación del algoritmo de hash SHA1

Tabla 4 - Ficheros fuente originales de iPXE reutilizados

La instalación y compilación del kernel de iPXE modificado se ha explicado en detalle en la sección 2.2.4. A continuación veremos las herramientas que se han empleado para la codificación de la solución.

5.3. DEPURACIÓN POR CABLE SERIE DE UNA MÁQUINA VIRTUAL

Durante toda la fase de implementación del prototipo ha sido fundamental la depuración del código que estaba siendo desarrollado. Este código, que se ejecuta en pre-boot, puede ser depurado de manera similar a como se depura el código de aplicaciones que se ejecutan sobre un sistema operativo. Sin embargo, el hecho de estar trabajando en pre-boot y con máquinas virtuales hace que la tarea de depuración sea un poco más compleja de lo habitual y requiera de una serie de pasos adicionales.

El kernel de iPXE está preparado de forma nativa para ser depurado a través de un puerto serie RS-232 o a través de una tarjeta de red. Pondremos el foco únicamente en el caso del puerto serie, ya que el caso de la tarjeta de red es similar y además presenta algunas limitaciones con respecto a la depuración a través de puerto serie. No entraremos a analizar dichas limitaciones.

La conexión ha de establecerse mediante un cable serie que conecte el equipo en el que se ejecuta el kernel de iPXE a depurar con el equipo en el que se ejecuta la herramienta de depuración. En adelante nos referiremos a estos equipos como el depurado y el depurador, respectivamente.

Como se ha mencionado en la sección 5.1, la implementación del prototipo ha sido realizada casi íntegramente utilizando máquinas virtuales de Virtual Box en lugar de máquinas físicas. Eso supone que tanto el puerto serie como el cable que conecta los equipos depurado y depurador son igualmente recursos virtualizados, es decir, no tienen entidad física. Las máquinas virtuales obviamente se ejecutan contenidas en una máquina física que las alberga, y que recibe el nombre de *host* o *anfitrión* aunque su independencia con ella a nivel lógico es total.

En este capítulo se describe cómo emular esta conexión entre el equipo depurador, que es un equipo físico, con el equipo virtual en el que corre el kernel de iPXE que se quiere depurar.

El primer paso es indicar al hipervisor de virtualización que queremos que la máquina virtual cuente con un puerto serie del tipo RS232. Esto se hace desde el menú *Preferencias* del administrador de máquinas virtuales. A continuación es necesario acceder al apartado *Puertos* e indicar que se desea habilitar un puerto serie. El puerto serie virtual es implementado en el equipo físico como un recurso de tipo pipe (tubería). Por tanto, es necesario especificar un path en el sistema de ficheros, así como un nombre de fichero para dicho pipe. En las siguientes imágenes se puede ver cómo se crea el puerto serie.

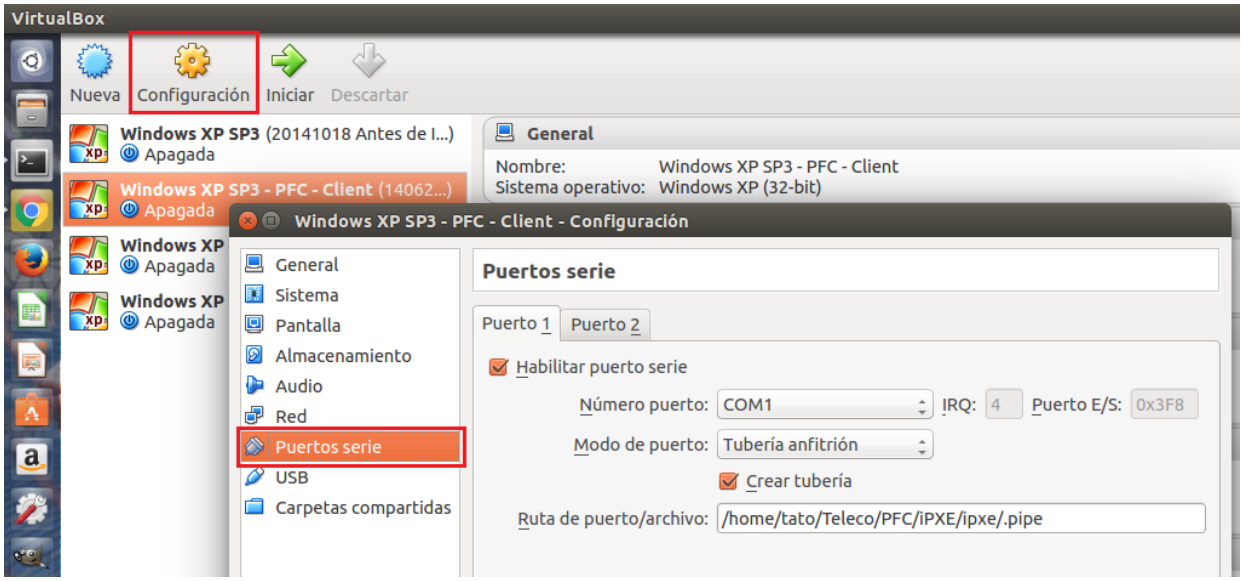


Ilustración 18 - Definición de un puerto serie en una máquina virtual

Una vez que disponemos de puerto serie virtualizado, el siguiente paso es conectar mediante un cable virtual el equipo depurador (físico) con el equipo depurado (virtual). Para emular dicha conexión utilizaremos el comando *socat.* Hay otras posibilidades, pero hemos elegido *socat* por ser muy sencillo de utilizar. Este comando recibe como parámetro el pipe creado en el apartado anterior y crea un manejador para el dispositivo en /dev/ptsX o bien /dev/ptyX, donde X es un número entero. La máquina virtual tiene que estar arrancada para que *socat* pueda enlazar el pipe (puerto serie de la máquina virtual) con el puerto manejador de dispositivo (puerto serie el equipo real). La siguiente imagen muestra como *socat* establece la conexión entre el pipe creado en el paso anterior y dispositivo /dev/pts11.

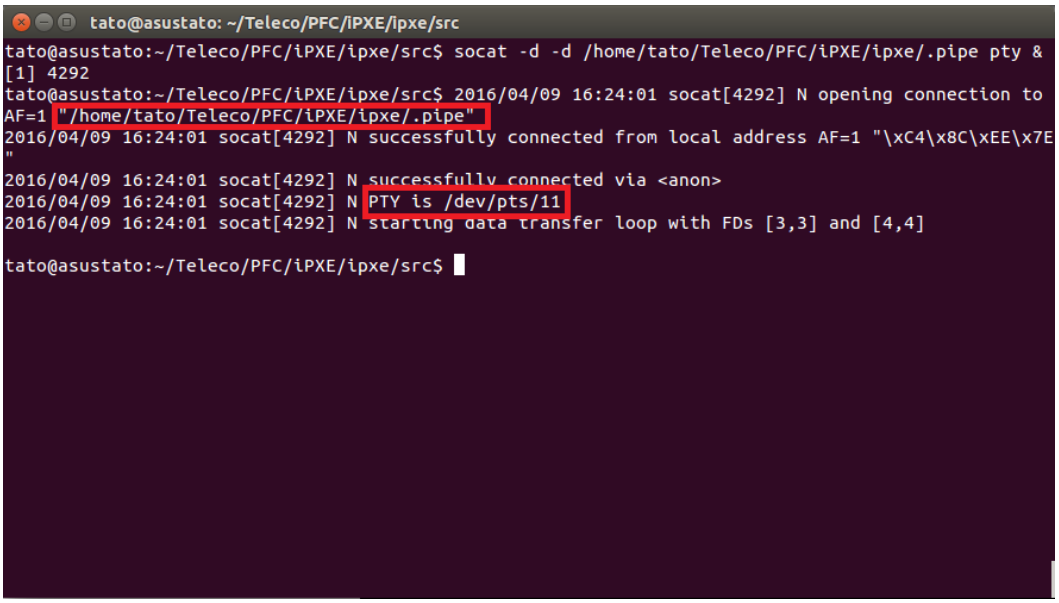


Ilustración 19 - Ejecución de socat para establecer conexión entre puertos serie

Una vez establecida la conexión virtual entre los puertos serie de los equipos depurador y depurado, necesitamos preparar el kernel de iPXE para ser depurado. Para ello solamente necesitamos introducir un

par de líneas de código en el fichero fuente en el que queremos establecer el punto de ruptura (punto en el cual el depurador se enlazará al programa en ejecución y detendrá su ejecución). Lo primero es incluir los ficheros de cabeceras gdbserial.h y gdbstub.h en el fichero fuente en el que se desea establecer el punto de ruptura, como se muestra en la siguiente imagen:

```
#include <ibc/ibehelper.h>
#include <ibc/ibetypes.h>
#include <ibc/ibcsha2.h>
#include <ibc/disk.h>
#include <ibc/util.h>
#include <ipxe/aes.h>
#include <ibc/point.h>
#include <unistd.h>

// For debugging purposes
#include <ipxe/gdbserial.h>
#include <ipxe/gdbstub.h>


/** @file
 *
 * GENERIC TCP/UDP IBC client
 *
 */

////AFZC: FEATURE ( FEATURE_PROTOCOL, "IBCPEER", DHCP_EB_FEATURE_IBCPEER, 1 );

/* IBCPEER-specific error codes */
#define EINVAL_BLKSIZE  __einfo_error ( EINFO_EINVAL_BLKSIZE )
```

Ilustración 20 - Includes necesarios para hacer debug con gdb

A continuación se colocará el punto de ruptura en el fichero simplemente insertando una llamada a la función gdbstub_start(), como se ilustra en la siguiente imagen:

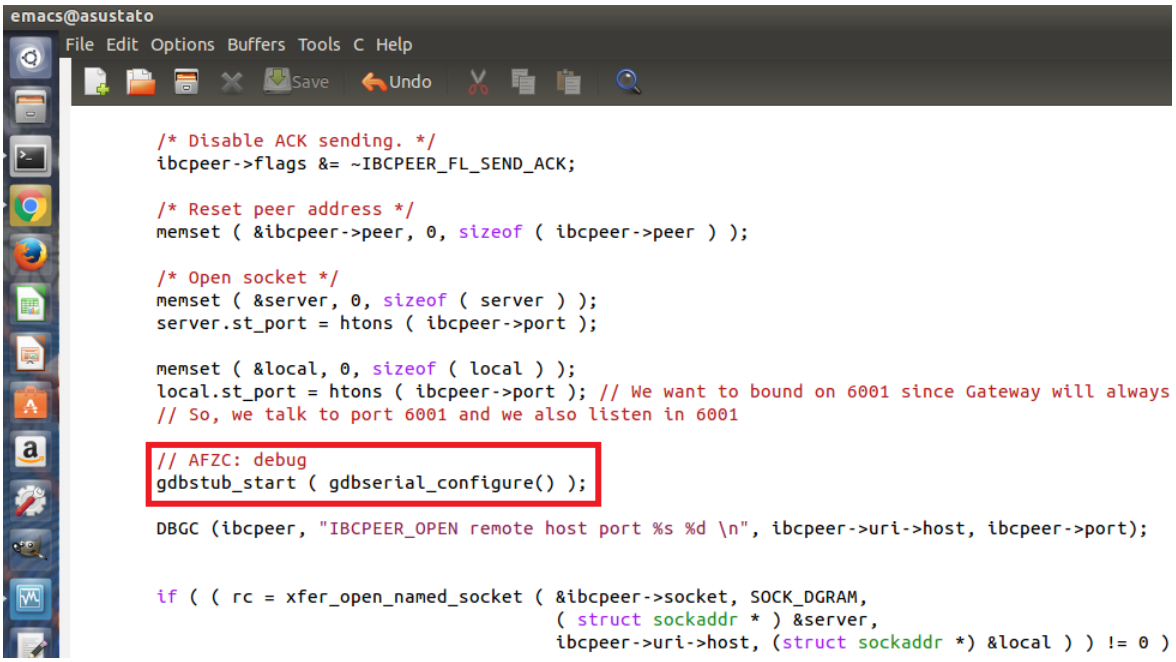


Ilustración 21 - Establecimiento del punto de ruptura para el depurador

A continuación se compilará el kernel de iPXE, tal y como se indicó en la sección 2.2.4.1.

En este punto ya estamos en disposición de lanzar el programa depurador gdb, y conectarnos al equipo virtual remoto para depurar el kernel. Al compilar el kernel obtendremos en el directorio de

binarios no solamente el ejecutable ipxe.lkrn, sino también el ejecutable con símbolos de depuración ipxe.lkrn.tmp. Será este el que le indicaremos al programa gdb que debe utilizar para la depuración.

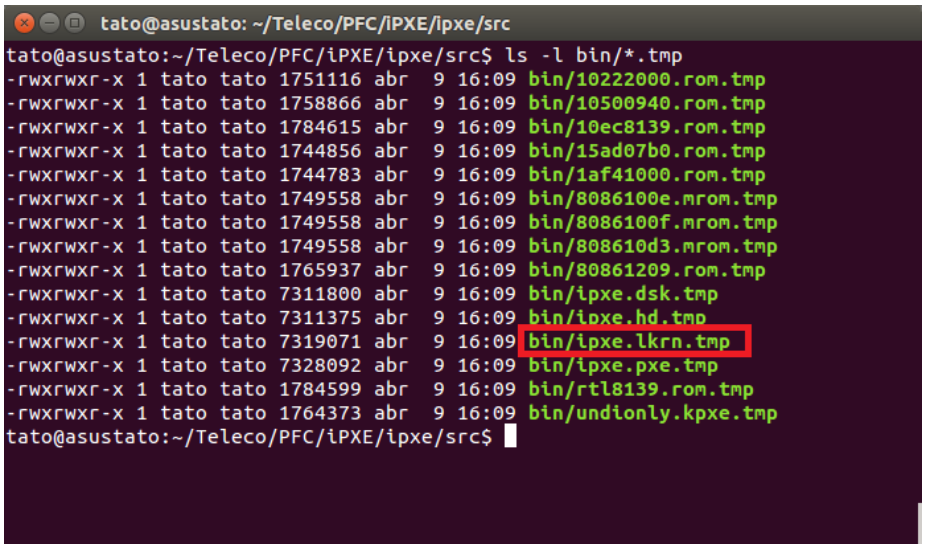


Ilustración 22 - Imagen del kernel iPXE con símbolos de depuración

El programa gdb ha de ser lanzado del siguiente modo:

```
]$ gdb bin/ipxe.lkrn.tmp
```

Una vez en la consola de depuración de gdb, se deberán establecer la velocidad en baudios del puerto serie y el descriptor de dispositivo del puerto serie en el equipo depurador. Para ello se ejecutarán las siguientes dos instrucciones:

```
gdb> set remote baud 9600
```

```
gdb> target remote /dev/ptyX
```

La siguiente imagen ilustra la ejecución del depurador. En este ejemplo se ha establecido el punto de depuración en la línea 201 del archivo fuente ibcpeer.c.

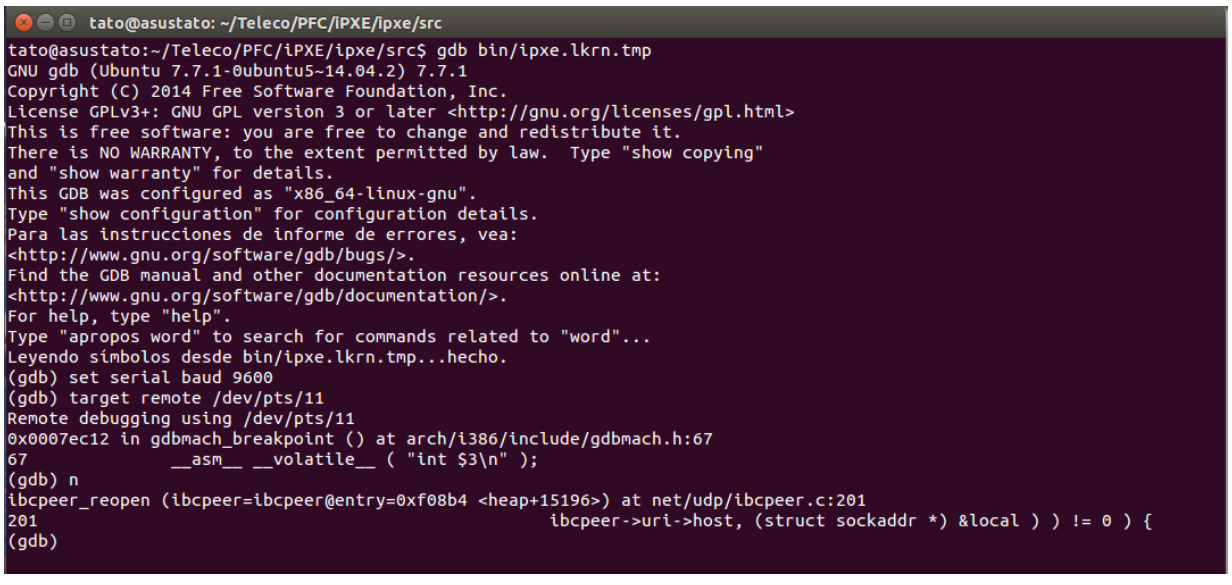


Ilustración 23 – Programa gdb en depuración del kernel de iPXE

Para finalizar veamos ahora el extremo depurado. La siguiente imagen muestra cómo el kernel de iPXE está a la espera de instrucciones del depurador para continuar su ejecución en el endpoint virtualizado. A partir de este punto la ejecución del kernel depende totalmente del programa depurador gdb.

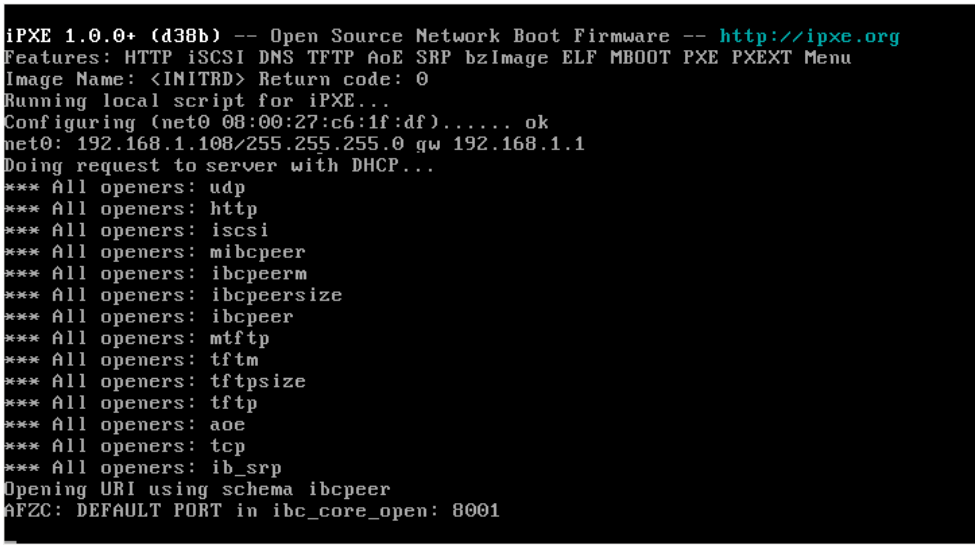


Ilustración 24 - Kernel de iPXE siendo depurado a la espera de instrucciones del depurador

5.4. ESCRITURA EN DISCO EN ENSAMBLADOR

Una de las limitaciones más importantes del kernel de iPXE es que no tiene soporte para el sistema de ficheros. Esto implica que no podemos acceder al disco duro a leer o escribir ficheros y por lo tanto no hay posibilidad de persistir información para utilizarla entre sucesivos arranques del sistema.

En la sección 4.2 se ha descrito cómo los equipos que pretenden hacer uso del cripto-sistema diseñado obtienen sus correspondientes claves privadas del Centro Generador de Claves. Estas claves están pensadas para tener una validez temporal de varios meses y por tanto no tiene sentido que los endpoints soliciten al Centro Generador de Claves una nueva clave cada vez que entren en funcionamiento. Además de ser computacionalmente costoso, presenta el inconveniente de que, si el Centro Generador no estuviera disponible por algún motivo, el endpoint no podría obtener su clave privada y por tanto no podría establecer comunicación alguna con otros equipos.

Por este motivo se han implementado unas sencillas funciones de lectura y escritura en disco duro. La implementación se basa en el uso de la interrupción 13. Esta interrupción permite leer o escribir sectores del disco duro utilizando el método de direccionamiento CHS (Cilindro-Cabeza-Sector).

La unidad básica de lectura o escritura es el sector (bloque de 512 bytes). Dado que la clave privada en el sistema diseñado es un punto de una curva elíptica, el tamaño de la clave privada es variable. Recordemos que un punto de la curva elíptica tiene dos coordenadas cuyo tamaño depende del campo finito subyacente a la curva. En la curva P-256 seleccionada como curva por defecto para el sistema implementado, las coordenadas del punto tienen una longitud de 256 bits (32 bytes). Esto significa que la clave privada ocupa 64 bytes, por lo que con un único sector tenemos capacidad de almacenamiento suficiente para almacenar la clave privada.

A continuación se presentan las funciones de lectura y escritura de la clave privada en ensamblador. Nótese que en ambas funciones es necesario traducir desde el espacio de direcciones en modo real al espacio de direcciones de usuario. Ambas funciones utilizarán el sector 60 del disco que es el que se ha decidido utilizar para almacenar la clave privada. Este sector se ha elegido arbitrariamente entre los sectores ocultos disponibles. Los denominados sectores ocultos son aquellos que conforman la primera pista del disco duro, es decir, los primeros 63 sectores del disco. Recordemos que el primer sector del disco duro recibe el nombre de Registro Maestro de Arranque y que almacena el código de bootstrap. El cargador de arranque GRUB4DOS que se instala en los endpoints ocupa en total los primeros catorce sectores del disco. Por tanto los sectores que van desde el decimoquinto sector hasta el sexagésimo tercero son sectores ocultos que tenemos a nuestra disposición para almacenar la clave privada.

Función de escritura a disco:

```
/*************************************************************/
/* Function: writeSecretToDisk                               */
/* Author: Angel F. Zato del Corral                          */
/* Date: 27-05-2015                                          */
/* Description:                                              */
/*    Write a secret (private key) into the hidden sectors   */
/*    of the hard disk. Sector written is the 60th           */
/*                                                           */
/*************************************************************/

int writeSecretToDisk (char * secret) {

  uint8_t status;
  int discard_b, discard_c, discard_d;
  int i;
  struct segoff address;
  unsigned char buffer[512];

  printf ("\n\nThe secret to store is %sh\n", secret);
  for (i=0; i<512; i++)
      buffer[i] = 0;  // Clear buffer content. All bytes set to 0x0

  memcpy (buffer,secret,strlen(secret));   // Copy private key to buffer

  /* Use INT 13, 03 to write sector from ES:BX to disk  */
  address.segment = 0;
  address.offset = 0x7c00;

  // Put buffer content in ES:BX so that interrupt 13h takes the bytes
  // from there and writes to the specified sector
  put_real( buffer, address.segment, address.offset );

  __asm__ __volatile__ ( REAL_CODE ( "pushw %%es\n\t"
                                     "pushl %%ebx\n\t"
                                     "popw %%bx\n\t"
                                     "popw %%es\n\t"
                                     "stc\n\t"
                                     "sti\n\t"
                                     "int $0x13\n\t"
                                     "sti\n\t" /* BIOS bugs */
                                     "jc 1f\n\t"
                                     "xorw %%ax, %%ax\n\t"
                                     "\n1:\n\t"
                                     "popw %%es\n\t" )
                                     : "=a" ( status ), "=b" ( discard_b ),
                                     "=c" ( discard_c ), "=d" ( discard_d )
                                     : "a" ( 0x0301 ), "b" ( address ),
                                     // One sector to write. Address is the
                                     // C variable buffer to write
                                     "c" ( 60 ), "d" ( 0x0080 ) );
```

```
  if ( status ) {
    DBG ( "INT13 drive 0x0080 could not write sectors (status %02x)\n", status );
    return -1;
  }

  return 0;

} /* end of function */
```

Función de lectura del disco:

```
/*************************************************************/
/* Function: readSecretFromDisk                              */
/* Author: Angel F. Zato del Corral                          */
/* Date: 27-05-2015                                          */
/* Description:                                              */
/*    Read the secret (private key) stored in the hidden     */
/*    sectors of the hard disk using BIOS interrupt 13h      */
/*************************************************************/

int readSecretFromDisk  (char * secret) {

   uint8_t status;
   int discard_b, discard_c, discard_d;
   uint16_t magic;
   unsigned char leido;
   uint16_t i;

   struct segoff address;

   /* Use INT 13, 02 to read the MBR */
   address.segment = 0;
   address.offset = 0x7c00;

   __asm__ __volatile__ ( REAL_CODE ( "pushw %%es\n\t"
                                      "pushl %%ebx\n\t"
                                      "popw %%bx\n\t"
                                      "popw %%es\n\t"
                                      "stc\n\t"
                                      "sti\n\t"
                                      "int $0x13\n\t"
                                       "sti\n\t" /* BIOS bugs */
                                      "jc 1f\n\t"
                                      "xorw %%ax, %%ax\n\t"
                                      "\n1:\n\t"
                                      "popw %%es\n\t" )
                                      : "=a" ( status ), "=b" ( discard_b ),
                                      "=c" ( discard_c ), "=d" ( discard_d )
                                      : "a" ( 0x0201 ), "b" ( address ),
                                      // AL = number of sectors to read
                                      "c" ( 60 ), "d" ( 0x0080 ) );
                                      // CL = sector to read
   if ( status ) {
       DBG ( "INT13 drive 0x0080 could not read MBR (status %02x)\n", status );
       return -1;
   }

   /* Check magic signature */
   get_real ( magic, address.segment,
              ( address.offset + offsetof ( struct master_boot_record, magic ) ) );

   if ( magic != INT13_MBR_MAGIC ) {
      DBG ( "INT13 drive 0x0080 does not contain a valid MBR\n");
     // return -1;
   }
```

```
        /* Get the 512 bytes of sector 60th of first track (the secret is there) */
        printf ("Reading secret from hidden sectors of disk...\n");
        for (i = 0; i < 512; i++) {
           /* get_real obtiene el valor que hay en la posicion de memoria
           segment:offset */

           get_real ( leido, address.segment, address.offset+i);
           printf ("0x%02x ", leido);
           secret[i] = leido;
        }
        return 0;
}
```

5.5. RESUMEN Y CONCLUSIONES

A lo largo del presente capítulo hemos visto los aspectos más importantes relacionados con la implementación de la solución.

La principal salida o producto tangible de este proyecto es la librería C, que implementa el cripto-sistema diseñado. La librería ha sido codificada dentro del proyecto original de iPXE, pues va a ser utilizada dentro de este kernel, pero se ha desarrollado de forma que puede ser extraída y utilizada fuera del kernel de iPXE.

Hemos enumerado y descrito brevemente los ficheros de código fuente más importantes de la solución. Todos ellos son ficheros que contienen código C y en algunos casos código ensamblador embebido.

Después hemos hecho un repaso de las herramientas utilizadas durante el desarrollo, no solo para la codificación en sí, sino también para la depuración y comprobaciones que eran necesarias para poder asegurar que la solución desarrollada cumple con su cometido.

Por último hemos desarrollado con algo más de detalle el método de depuración de software que se ejecuta en pre-boot en el contexto de una máquina virtual por considerarlo algo novedoso que no suele verse habitualmente en proyectos de desarrollo software. Por razones similares nos hemos detenido también a detallar la implementación de un acceso básico a disco codificado en lenguaje ensamblador.

La finalización de este apartado da pie de forma natural al siguiente capítulo, en el que veremos cómo se han definido y ejecutado las pruebas necesarias para comprobar que la solución implementada hace lo que tiene que hacer, no hace lo que no tiene que hacer y se comporta de forma estable sin producir daños inesperados al equipo en el que se ejecuta.

6. VALIDACIÓN

Una vez finalizada la implementación e integración (dentro del proyecto de iPXE) de la librería criptográfica es necesario comprobar que todo funciona como se espera y que no se producen efectos colaterales no deseados. Para ello son necesarias la definición y ejecución de pruebas que realicen validaciones a distintos niveles. Las pruebas se han divido en los siguientes bloques:

- Pruebas unitarias de operaciones criptográficas
- Pruebas de integración en el kernel de iPXE
- Pruebas de instalación
- Pruebas de integración del sistema completo

6.1. PRUEBAS UNITARIAS DE OPERACIONES CRIPTOGRÁFICAS

El objetivo de estas pruebas es asegurar que la codificación de las operaciones y algoritmos criptográficos producen los resultados que se esperan. Para ello se ha desarrollado un pequeño programa (test_ibc_cryptography.c) en el que se han incluido una serie de operaciones criptográficas cuyo resultado se conoce y por tanto se puede comprobar la validez de los cálculos.

Este programa fija los valores de:

- s (secreto maestro)
- P (punto generador en la curva)
- a y b (coeficientes que definen la curva elíptica)
- Q (punto genérico en la curva)
- r (valor numérico que se utilizará como si se tratara de un número aleatorio). Es necesario fijar el valor para que los resultados sean deterministas y se puedan efectuar las comprobaciones necesarias.

Con estos valores establecidos el programa realiza y comprueba (haciendo uso de la librería desarrollada) que son correctos los siguientes cálculos:

- 2P (Doblaje de P)
- P + Q (Suma de los puntos P y Q)
- mapIdentity (“CADENA_DE_TEXTO_PRESTABLECIDA”)
- sP (multiplicación escalar del valor s por el punto P). Nótese que este valor sería la clave pública del cripto-sistema y P el punto generador.
- Producción de la firma ECDSA para el valor numérico r interpretado como el mensaje a firmar.
- Validación de la firma ECDSA producida para el valor numérico (mensaje) r.
- e(rP,sQ). Emparejamiento bilineal.

Este conjunto de operaciones se considera suficiente para dar por cubiertas todas las operaciones y algoritmos implicados en la solución. De hecho, únicamente con las operaciones de cálculo y verificación de firma ECDSA y de emparejamiento bilineal habría sido suficiente, ya que cada una de esas operaciones conlleva un número muy elevado de operaciones básicas, entre las que se incluyen todas las anteriores.

El cuerpo principal de este programa es un bucle definido para realizar mil iteraciones. Esto significa que las comprobaciones anteriores se realizan mil veces. Puesto que todos los valores están fijados el objetivo principal de repetir los mismos cálculos un millar de veces es poner a prueba la codificación en lo que respecta al uso de memoria dinámica y al manejo de punteros ya que en cada iteración se van a producir reservas y liberaciones de memoria y una gran cantidad de cálculos que implican manejo de punteros. Si hubiera algún tipo de error de codificación relacionado con estos aspectos, el programa acabaría produciendo una violación de segmento.

Por cada uno de los cálculos el programa produce una salida por pantalla de OK o NOK (no OK). Obviamente la salida OK corresponde al caso en el que el cálculo produce el valor esperado, mientras que la salida NOK se produce cuando el valor obtenido no es el esperado. Para no tener que estar atentos a la salida que se produce por pantalla, se ha trabajado redirigiendo la salida a un fichero, sobre el que posteriormente se ha revisado que todos los cálculos han devuelto OK.

Este bloque de pruebas se ha llevado a cabo en el equipo de desarrollo, ya que el único objetivo que tiene es comprobar el funcionamiento correcto de las funciones y algoritmos implementados en la librería.

A continuación se muestra una captura de pantalla del programa *test_ibc_cryptography* en ejecución.

Ilustración 25 - Programa de validación de las funciones criptográficas en ejecución

6.2. PRUEBAS DE INTEGRACIÓN EN EL KERNEL DE iPXE

Si en el apartado anterior hemos visto cómo se comprueba la validez de las funciones y algoritmos criptográficos implementados de forma general, en este bloque de pruebas vamos un paso más allá y comprobamos que el funcionamiento de esos mismos algoritmos y funciones, cuando se ejecutan en pre-boot dentro del kernel de iPXE, es igualmente correcto. Pero, sobre todo, verificamos que no hay problemas asociados a la gestión de recursos (memoria principalmente) en el contexto de pre-boot en el que, como ya se ha indicado anteriormente, se funciona en modo real y en el que además existen importantes restricciones en cuanto al uso de memoria en comparación con la ejecución en modo protegido (cuando existe un sistema operativo cargado y es este el que da soporte a la ejecución del programa).

Por tanto esta prueba esencialmente consiste en trasladar el código del programa *test_ibc_cryptography* mencionado en el apartado anterior al kernel de iPXE.

Para ello, simplemente se ha hecho una copia del código del programa (todo él se ejecuta dentro una función) dentro de la función *main* del kernel de iPXE. La función main del kernel de iPXE se encuentra en el fichero ./src/core/init.c del directorio de fuentes del proyecto.

Al igual que en la prueba anterior los cálculos se van a realizar un millar de veces. En este caso, en el que el programa se ejecuta en pre-boot, los errores de codificación relativos a la gestión de memoria y al manejo de punteros tienen un impacto mucho más grave y es fundamental tener la certeza de que no se han cometido errores de ese tipo. Recordemos que la asignación de memoria para el programa en ejecución, así como el tamaño de la propia pila de llamadas, es mucho menor, por lo que un pequeño *leak* puede conducir rápidamente al agotamiento de la memoria disponible para el programa, traduciéndose en un malfuncionamiento o en la detención brusca e inesperada de su ejecución. Por su parte, cuando un programa que se ejecuta en modo protegido produce un error de violación de memoria, el sistema operativo subyacente se encarga de finalizar el programa de forma controlada indicando la causa del problema y protegiendo el resto de los programas y el propio sistema operativo de los efectos adversos que esa mala gestión de la memoria podría haber ocasionado. Cuando se produce un error de memoria en modo real, las consecuencias pueden ser muy graves, llegando incluso a dejar la máquina completamente colgada, sin posibilidad alguna de recuperación. Por este motivo la realización de estas pruebas en pre-boot juega un papel fundamental de cara a asegurar la estabilidad de la solución desarrollada.

En la siguiente imagen se pueden ver nuevamente las comprobaciones que hace el programa *test_ibc_cryptography*, pero esta vez ejecutándose en el contexto del kernel de iPXE, es decir, ejecutándose en pre-boot en modo real.

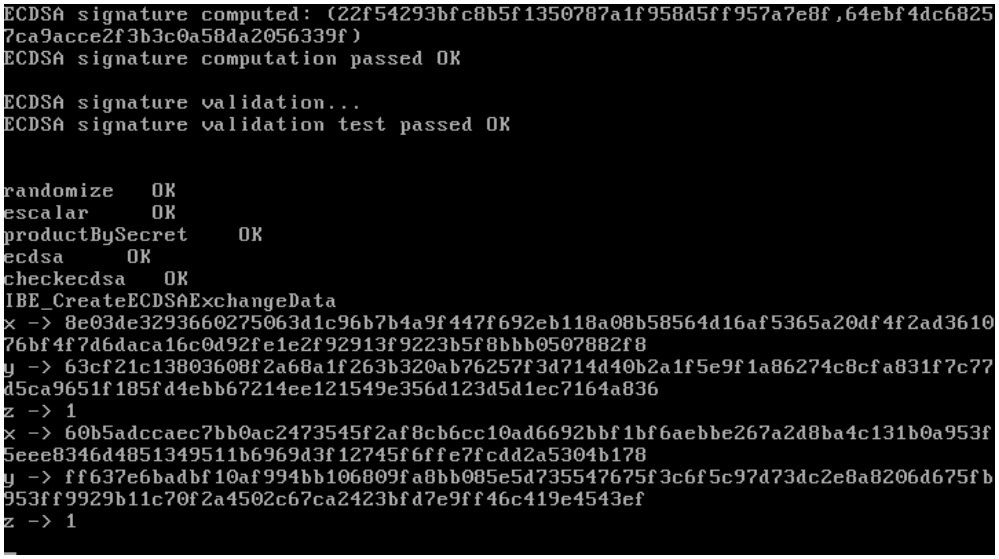


Ilustración 26 - Programa de validación de las funciones criptográficas en ejecución en pre-boot

6.3. PRUEBAS DE INSTALACIÓN

Una vez que se hemos validado el correcto funcionamiento de los algoritmos y primitivas criptográficas en tiempo de pre-boot, el siguiente paso en la cadena de validación es comprobar que la instalación del prototipo en equipos reales es factible y que no produce ningún problema a los equipos.

Estas pruebas consisten por tanto en seleccionar un conjunto de equipos lo más heterogéneo posible e instalar el prototipo de comunicaciones seguras en pre-boot, lo que en esencia consiste en instalar los siguientes componentes:

- Código de bootstrap en el MBR de GRUB4DOS
- Ficheros menu.lst y grldr en el disco duro del equipo
- Kernel de iPXE (fichero ipxe.lkrnl)

La instalación de estos componentes ya se ha explicado con detalle en los distintos subapartados de la sección 2.2.4.

La realización de estas pruebas de instalación resulta muy sencilla desde el punto de vista de las tareas a realizar. Sin embargo, resulta complicada en el sentido de que no es fácil conseguir equipos físicos en los que probar incluso en una empresa grande como en la que se ha realizado el proyecto. La instalación de este tipo de software supone acciones muy intrusivas para los equipos y el riesgo de dejar las máquinas inoperativas es, cuando menos, alto. Por ese motivo es necesario realizar copias de seguridad del disco de los equipos antes de ejecutar la prueba de instalación, lo que conlleva largos períodos de espera antes de poder realizar una prueba que por sí misma no lleva más de diez minutos.

A continuación se enumeran los distintos equipos (PCs de sobremesa, portátiles y kioscos) sobre los que se ha validado la instalación del prototipo de comunicaciones segura. La tabla ofrece información sobre el fabricante, la geometría, el tamaño y la velocidad del disco duro del equipo, ya que es el factor diferencial que puede dar lugar a problemas. La geometría del disco se define por una pareja de valores: el número de pistas por cilindro y el número sectores por pista.

MODELO PC/KIOSCO	FABRICANTE DISCO	GEOMETRÍA DISCO	TAMAÑO DISCO	VELOCIDAD DISCO
HP 6000 Series	SEAGATE	255x63	500 GB	7.200 RPM
Talaris CashStar 9110	SEAGATE	255x63	80 GB	5.400 RPM
Wincor ProCash 1500	WESTERN DIGITAL	60x12	40 GB	5.400 RPM
Toshiba Tecra M5	WESTERN DIGITAL	255x63	100 GB	5.400 RPM
KEBA KePlus R6se	HITACHI	255x63	250GB	7.200 RPM
NCR 5886	WESTERN DIGITAL	255x63	2 GB	5.200 RPM
NCR 6622	SEAGATE	60X12	70 GB	7.200 RPM
HP Compaq Pro6300	SEAGATE	255x63	128 GB	7.200 RPM
Diebold Opteva 522	SEAGATE	255x63	320 GB	5.400 RPM
Diebold Opteva 520	HITACHI	30x222	120 GB	5.400 RPM

Ilustración 27 - Listado de modelos en los que se ha instalado el prototipo

Lógicamente, la prueba se da por satisfactoria únicamente cuando el cargador de arranque GRUB4DOS queda correctamente instalado y tanto el kernel de iPXE como el sistema operativo original de equipos pueden ser arrancados. Por tanto, por cada prueba de instalación realizada, se comprueba que el equipo inicia correctamente el kernel de iPXE y, en otro arranque independiente, el sistema operativo Windows original.

Algunos de estos equipos presentaron en primera instancia resultados insatisfactorios a la hora de instalar el prototipo. Posteriormente se pudo comprobar que todos ellos presentaban errores a nivel lógico

o físico en el disco duro o presentaban una geometría que no era conforme a la especificación del fabricante del disco. Una vez reparados los daños el prototipo pudo instalarse sin problemas.

6.4. PRUEBAS DE INTEGRACIÓN Y DEL SISTEMA

Estas pruebas suponen el último eslabón en la cadena de validación del prototipo. Se trata de evaluar el funcionamiento de la solución diseñada (con todas sus piezas ensambladas) en un entorno real, con endpoints reales y estableciendo comunicaciones con servidores reales.

Para la realización de estas pruebas se han utilizado los siguientes recursos y componentes:

- Dos endpoints reales. Un endpoint del fabricante Talaris modelo CashStar 9110 con sistema operativo Windows XP 32 bits y un endpoint del fabricante Wincor modelo ProCash 1500 con sistema operativo Windows 7 Professional de 32 bits.

- Un equipo PC que actúa de servidor genérico con el que los endpoints se conectan. El servidor genérico está implementado en Java y ha sido desarrollado por otro equipo de personas de la empresa, pero utiliza la librería criptográfica desarrollada en este proyecto. El equipo es un HP 6000 series equipado con un Windows 7 Professional de 64 bits. En este mismo equipo, aunque como un proceso independiente, también se ejecuta el Centro Generador de Claves, que está implementado nuevamente como una aplicación Java con base en la librería criptográfica desarrollada. El motivo de ejecutar el Centro Generador de Claves y el servidor genérico en el mismo equipo físico es simplemente evitar el uso de un equipo físico adicional.

- La red de comunicaciones que interconecta todos los equipos es una red de área local TCP/IP. En dicha red existe un servidor DHCP que proporciona direcciones IP en la red 192.168.169.0/24 a aquellos equipos que lo solicitan. Los dos endpoints cuentan con tarjetas Ethernet 10/100 Mbps, mientras que el equipo servidor cuenta con una tarjeta Ethernet de 1Gbps, que es la configuración más común en los entornos reales de endpoints.

En ambos kioscos se instala el prototipo de comunicaciones seguras y se configura un script local de iPXE para que conecten con el servidor genérico tan pronto como obtengan una dirección IP del servidor DHCP.

Por su parte el servidor genérico está programado para que, tan pronto como la conexión segura sea establecida con éxito, envíe el texto de control “OK123” (que obviamente se transmitirá cifrado hacia el endpoint). El kernel de iPXE del endpoint se programa para esta prueba de modo que, si recibe el texto “OK123” del servidor, da por finalizada con éxito el establecimiento de la conexión segura y transfiere el control de la cadena de arranque al cargador de Windows para que sea este quien inicie el sistema. En caso de no recibir nada o recibir cualquier otro valor, el kernel de iPXE detendrá su ejecución y mostrará un mensaje de error por pantalla. Recuérdese que para que el endpoint pueda establecer comunicaciones seguras con el servidor (o con cualquier otro equipo remoto en el cripto-sistema), primeramente establece conexión con el Centro Generador de Claves para recuperar su clave privada, tal y como se describió en la sección 4.2. Una vez obtenida su clave privada, establecerá la comunicación segura con el servidor genérico del modo en que se describió en la sección 4.3. El kernel de iPXE en el endpoint se programa de forma especial para esta prueba, de forma que, aunque la clave privada se almacena en los sectores ocultos del disco duro, se solicita una nueva clave al Centro Generador de Claves en cada arranque.

Por último, los sistemas operativos Windows de los endpoints han sido configurados con una tarea periódica que reinicia los equipos tan pronto como la carga del sistema operativo Windows ha finalizado. Con este reinicio se pretende encerrar al endpoint en un bucle en el que tiene que repetir todo el proceso de establecer conexión segura con el servidor genérico.

El siguiente esquema ilustra cómo se ejecuta esta prueba de integración y del sistema en entorno real:

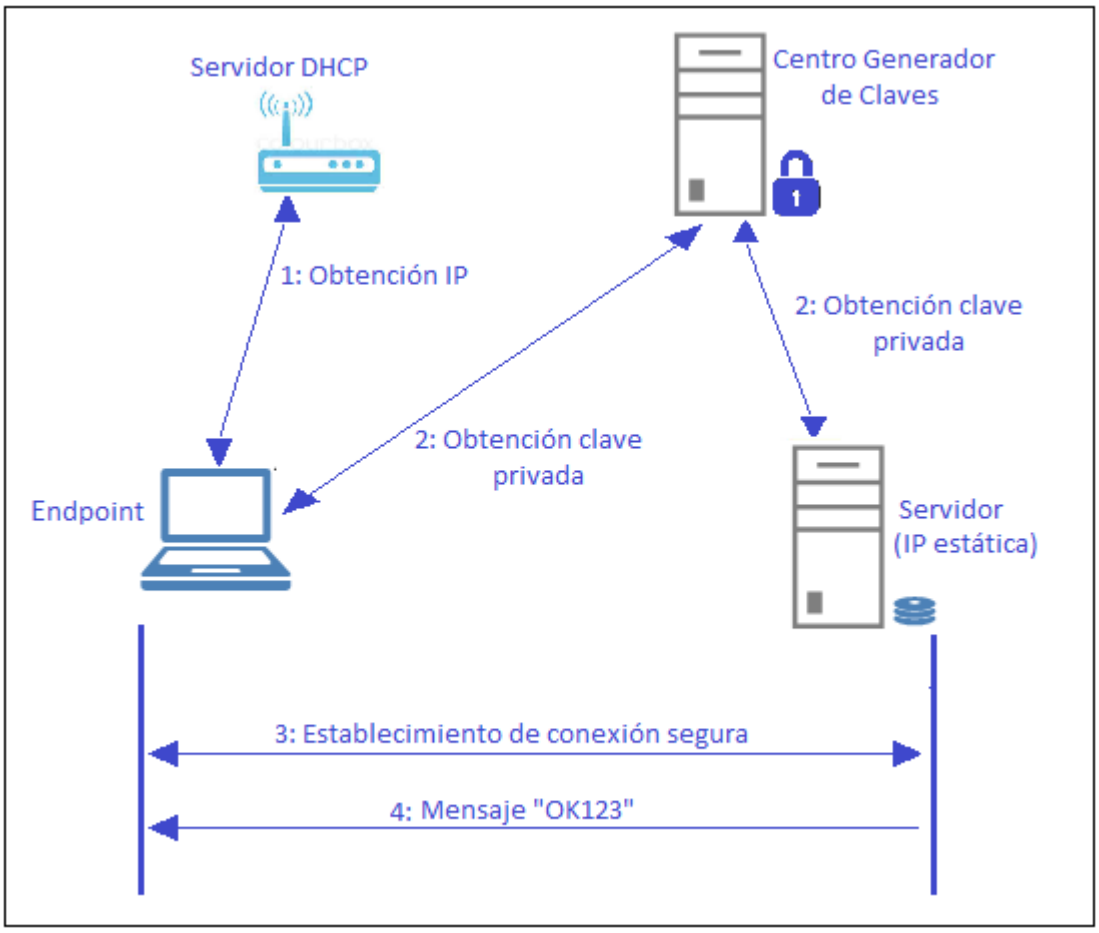


Ilustración 28 - Esquema de prueba de integración

En el paso 1 el endpoint obtiene una dirección IP del servidor DHCP de la red. El Servidor (genérico) se configura con dirección IP estática (que es lo más común en los entornos de endpoints) y por lo tanto no tiene necesidad de acudir al servidor DHCP para obtener una dirección IP.

En el paso 2 tanto el Servidor como el endpoint acuden al Centro Generador de Claves a obtener su clave privada.

En el paso 3 el endpoint establece la conexión segura con el Servidor. Si esta queda establecida correctamente, ambos equipos han autenticado al otro extremo y cuentan con un canal cifrado para el intercambio confidencial de datos.

En el paso 4 el servidor envía un mensaje de control con el contenido “OK123” que el endpoint utilizará como testigo para dar por finalizada con éxito la prueba y reiniciarse, volviendo a comenzar la prueba desde el paso 1.

La realización de esta prueba de integración requiere la observación presencial de la misma. Es necesario estar delante del endpoint para observar si todo el proceso transcurre según lo esperado (en cuyo caso el equipo se reinicia) o si por el contrario la prueba falla y el kernel de iPXE queda detenido, mostrando algún mensaje de error por pantalla.

6.5. RESUMEN Y CONCLUSIONES

En este capítulo hemos visto cómo se valida el prototipo desarrollado mediante distintas pruebas. Comenzamos por pruebas unitarias para comprobar que los algoritmos criptográficos han sido codificados correctamente y producen los resultados esperados.

Posteriormente trasladamos las pruebas relacionadas con la criptografía al contexto de ejecución de pre-boot con el objetivo primordial de comprobar que no se han cometido errores en la gestión de la memoria o el manejo de punteros, ya que los errores de este tipo cuando se opera en modo real pueden tener consecuencias muy graves para los endpoints.

A continuación se valida la instalación de los componentes del prototipo en un conjunto heterogéneo de equipos que pretende ser una muestra representativa de los endpoints que se van a encontrar posteriormente en el mundo real.

Por último realizamos pruebas de integración, uniendo todas las piezas y poniéndolas a funcionar en conjunto.

Es importante resaltar que en las redes de endpoints que encontramos en el mundo real el número de modelos y combinaciones de componentes hardware es realmente elevado y por tanto, aunque se ha probado en una buena cantidad de equipos distintos, se considera necesario ampliar las pruebas del sistema a un número mucho mayor de equipos antes de poder pasar a la fase de producción y comercialización.

7. CONCLUSIONES

7.1. RESUMEN

Se acerca la hora de poner fin a la memoria. Antes de exponer las últimas conclusiones hagamos un repaso final de lo que ha dado de sí el proyecto y en consecuencia el presente documento.

Iniciamos el proyecto enunciando el problema para el que se pretendía encontrar solución. Este problema no es otro que el de buscar una solución adecuada para establecer comunicaciones seguras en tiempo de pre-boot para una serie de equipos terminales que denominamos endpoints y que se caracterizan por tener una capacidad computacional limitada y una serie de dispositivos periféricos de alto valor.

Los primeros pasos del proyecto los dedicamos al estudio de las comunicaciones en pre-boot, que son aquellas que tienen lugar antes de que se produzca la carga del sistema operativo. Revisamos qué son y para qué se usan, lo que ineludiblemente nos condujo al estudio del estándar PXE. A continuación nos dedicamos a estudiar el estado del arte y las alternativas que existen para proporcionar un soporte de comunicaciones en pre-boot que dé cobertura al mayor número de endpoints posibles. El estudio nos llevó a la selección del proyecto de código abierto iPXE como solución para dotar a los endpoints del mejor y más adecuado soporte para este tipo de comunicaciones tan particulares.

Una vez resuelto el episodio del soporte de comunicaciones en pre-boot, nos adentramos en la misión de securizar dichas comunicaciones. Fuertemente condicionados por el requisito de no utilizar certificados digitales, nuestra búsqueda nos condujo a un tipo de criptografía de clave pública no demasiado extendido que recibe el nombre de Criptografía Basada en Identidad, y que tiene la particularidad de que la clave pública de un determinado individuo se vincula a este de forma natural y no porque un tercero (en quien todos confían) emita un certificado que asegure o garantice el vínculo. La Criptografía Basada en Identidad se apoya en unas estructuras algebraicas denominadas Curvas Elípticas. Mientras que estas últimas están perfectamente estandarizadas y presentes en multitud de aplicaciones y sistemas, encontramos que para la Criptografía Basada en Identidad no sucede lo mismo, por lo que actualmente se considera una tecnología minoritaria. Aun con este inconveniente de la falta de estandarización jugando en su contra, resulta perfecta para nuestro objetivo. Dedicamos varias secciones de la memoria a realizar una introducción sobre ciertos aspectos de la criptografía moderna con el objetivo de contar con la base necesaria para comprender cómo funciona la Criptografía Basada en Identidad, que a su vez se sustenta en la Criptografía de Curva Elíptica.

Una vez sentados los conceptos diseñamos una solución que permite a nuestros endpoints establecer comunicaciones seguras con otros endpoints o con servidores externos. La solución encontrada da lugar a un sistema de tres fases: Establecimiento de los parámetros del propio sistema, Obtención de Claves Privadas proporcionadas por un Centro Generador de Claves y finalmente Establecimiento de la Conexión mutuamente autenticada y cifrada.

Basados en el diseño, el siguiente paso fue desarrollar la solución. La implementación da lugar a una librería codificada en C (y parcialmente en ensamblador) que proporciona todas las primitivas y algoritmos que sustentan el cripto-sistema basado en identidad. En torno a dicha librería se construye un prototipo completamente funcional para demostrar el funcionamiento del cripto-sistema diseñado.

Finalmente, se valida el sistema construido ejecutando una serie de pruebas diseñadas para comprobar la validez del sistema además de la estabilidad y robustez del mismo.

7.2. CONCLUSIONES FINALES

A lo largo de los distintos capítulos se han ido anticipando una serie conclusiones directamente relacionadas con dichos apartados. A continuación se expone otro conjunto de conclusiones de carácter general obtenidas tras la finalización del presente proyecto.

Las comunicaciones en pre-boot aparecen en el mundo real de la tecnología en mucha menor medida que las comunicaciones que tienen lugar con posterioridad a la carga del sistema operativo y quedan relegadas a un número reducido de aplicaciones y sistemas. Por ese motivo no existen muchas implementaciones y en ellas prevalecen otras características por encima de la seguridad. Aun así existen una serie de opciones interesantes que, al ser de código abierto, pueden servir como punto de partida para necesidades heterogéneas.

El desarrollo de software que se ejecuta en tiempo de pre-boot es realmente complicado. Se debe trabajar en modo real, sin la ayuda y la protección que habitualmente presta el sistema operativo, haciendo uso de lenguajes de bajo nivel para la codificación, llegando incluso al ensamblador, y recurriendo al uso de funciones de muy bajo nivel como son las interrupciones del procesador. La depuración del código diseñado para su ejecución en pre-boot es también una tarea compleja y la cantidad de herramientas de las que dispone el programador es considerablemente menor. Además, cualquier problema que se presente en tiempo de pre-boot puede dejar las máquinas completamente inoperativas por lo que en el desarrollo de este tipo de programas deben primar la robustez y la estabilidad por encima de todo.

Tal y como se ha expuesto, para realizar la instalación de una solución como el prototipo desarrollado es necesario acometer acciones que son consideradas muy intrusivas para el endpoint, como por ejemplo la instalación de un cargador de arranque en el MBR del disco. Por otra parte, el desconocimiento generalizado que existe al respecto de soluciones que funcionan en tiempo de pre-boot, al margen de un sistema operativo de los ampliamente extendidos y conocidos, hace que se las mire con cierto recelo y dificulte su aceptación e implantación en muchos sistemas. Además, y como se ha indicado anteriormente, cuando un programa que se ejecuta en pre-boot sufre algún tipo de problema, sea de la naturaleza que sea, el impacto puede ser muy grave, puesto que la máquina puede quedar totalmente colgada y requerir de intervención presencial para ponerla nuevamente en funcionamiento. Obviamente, en un entorno desatendido por naturaleza como es una red de endpoints, esto supone un riesgo considerable y la fiabilidad y estabilidad de la solución de pre-boot deben ser máximas.

Durante la fase de validación del prototipo se ha observado que el estado de los discos duros puede jugar un papel fundamental en la instalación de los componentes de la solución. En concreto se ha detectado que cualquier tipo de daño a nivel lógico (tabla de particiones inconsistente, geometría no conforme a la especificación del fabricante, sistema de archivos dañado, etc.) o físico (daños producidos en las superficies magnéticas) produce malfuncionamientos en el cargador de arranque instalado. Se considera conveniente, por tanto, realizar chequeos del estado del disco antes de proceder a la instalación de los componentes de la solución. En caso de encontrar cualquier tipo de defecto, se debe abortar la instalación y posponerla hasta que los problemas se hayan solventado. Existen muchas aplicaciones de diagnóstico de discos duros, pero, parece lógico que en el caso concreto de los endpoints bajo estudio, que

cuentan con un sistema operativo Windows se utilice la aplicación CHKDSK (Check Disk), que estos sistemas incorporan de serie y cuya alta eficacia ha sido contrastada durante años.

La criptografía moderna engloba conceptos como firmas o certificados digitales, cifrado, autenticación, etc. Todos estos conceptos de alto nivel son relativamente sencillos de comprender. Sin embargo, cuando bajamos a analizar las matemáticas de algunos algoritmos (como los que subyacen a los emparejamientos bilineales), la complejidad se hace mucho mayor, hasta el punto de que son necesarios profundos conocimientos de álgebra.

A pesar del impulso recibido con la aparición de los emparejamientos bilineales, la Criptografía Basada en Identidad está aún en un estadio temprano y no hay estandarización para ella por parte de los principales organismos reguladores. Existen algunos proyectos en curso para la estandarización de la tecnología, pero avanzan muy despacio. Es precisamente esta falta de estandarización la que tiene como consecuencia que estos sistemas no tengan penetración alguna, o esta sea casi despreciable, en aplicaciones comerciales.

Por su parte, la tecnología de curva elíptica está sólidamente implantada y presente en innumerables aplicaciones. El NIST, por ejemplo, ofrece una serie de curvas y puntos generadores para ser empleados en aplicaciones de seguridad. También encontramos soporte para algoritmos de firma y de cifrado basados en curvas elípticas en los navegadores web o en la misma Máquina Virtual de Java, aplicaciones que utilizamos a diario.

No puedo finalizar el apartado de conclusiones sin mencionar uno de los aspectos que más ha llamado mi atención durante la realización de este proyecto. Resulta notoriamente curioso cómo se emplean frecuentemente expresiones del tipo “se considera que”, “equivale aproximadamente a” o “es comúnmente aceptado que” a la hora de establecer relaciones de equivalencia o de comparación entre distintos algoritmos criptográficos. El uso de estas expresiones, ciertamente ambiguas, se puede encontrar incluso en publicaciones de organismos de referencia en esta disciplina, como pueden ser el NIST o la NSA. Obviamente, cuando se emplean este tipo de expresiones, se hace intencionadamente y no podemos concluir otra cosa que no sea el hecho de que no existe una forma puramente analítica para comparar o demostrar la bondad, robustez o eficiencia de determinados algoritmos frente a otros.

7.3. TRABAJO FUTURO

A continuación se muestras aquellos puntos identificados como susceptibles de mejora o que deberán ser revisados en mayor o menor medida antes de pasar a la productización del sistema diseñado:

- Uno de los primeros cambios a realizar es la implementación del Centro Generador de Claves en un hardware criptográfico tipo HSM. Este tipo de sistema constituye el soporte físico ideal para almacenar el secreto maestro de forma segura, además de que, al tratarse de una implementación en hardware, el tiempo de cómputo de las operaciones con números grandes se ve significativamente reducido.

- Hemos visto cómo el Centro Generador de Claves asigna claves privadas a los usuarios del sistema. Otra de las ventajas (que no hemos visto) de los sistemas criptográficos basados en identidad es

que resulta relativamente sencillo asignar a las claves privadas caducidad temporal a modo de fecha de expiración de los certificados digitales.

- Aunque el almacenamiento de la clave privada en los sectores ocultos del disco duro puede ser una solución razonablemente válida para los endpoints objeto de estudio, a medida que el hardware de estos vaya siendo remplazado por equipamientos más modernos, será posible hacer uso del TPM, que es un componente especialmente diseñado para el almacenamiento seguro de claves criptográficos entre otras funciones.

- En distintas partes del proyecto se recurre al algoritmo de cifrado simétrico AES-256 para garantizar la confidencialidad de los datos intercambiados. En concreto, se ha usado AES en el modo de encadenamiento de bloques cifrados (CBC). Este algoritmo requiere dos parámetros de entrada: la clave y el vector de inicialización. En todo momento hemos detallado cómo se construye la clave, pero no hemos hecho referencia al vector de inicialización. Esto es debido a que se ha utilizado como vector de inicialización el mismo valor que para la clave. Esta opción no es la más recomendable, y será una de las correcciones que se debería realizar en el corto plazo.

- Todo el análisis que se ha realizado sobre la secuencia de arranque, el direccionamiento CHS, el Registro Maestro de Arranque o MBR, los sectores ocultos, etc., se ha hecho asumiendo que el disco duro de los endpoints está particionado según el esquema MBR. Este supuesto es perfectamente válido en la actualidad, pero con el paso del tiempo el particionamiento MBR será sustituido paulatinamente por GPT y por tanto será necesario revisar tanto la instalación del cargador de múltiples sistemas operativos como la escritura en los sectores ocultos del disco.

8. BIBLIOGRAFÍA/REFERENCIAS

[1] Preboot Execution Environment (PXE) Specification

http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf

[2] PXELinux project:

http://www.syslinux.org/wiki/index.php/PXELINUX

[3] Certificate Revocation:

http://news.netcraft.com/archives/2013/05/13/how-certificate-revocation-doesnt-work-in-practice.html

[4] Digital Certificates and SSL

http://technet.microsoft.com/en-us/library/dd351044%28v=exchg.150%29.aspx

[5] Dynamic Host Configuration Protocol (DHCP) Options for the Intel Preboot eXecution Environment (PXE)

https://tools.ietf.org/html/rfc4578

[6] Method to authenticate clients and hosts to provide secure network boot

https://www.google.com/patents/US7299354?dq=Patent+US7299354&hl=en&sa=X&ei=RVOZVO7hDcruUP63grAL&ved=0CCEQ6AEwAA

[7] An Introduction to Identity-based Cryptography
https://courses.cs.washington.edu/courses/csep590/06wi/finalprojects/youngblood_csep590tu_final_paper.pdf

[8] Identity-Based Encryption from the Weil Pairing

https://www.iacr.org/archive/crypto2001/21390212.pdf

[9] Master thesis on Paring-Based Cryptography by Martijn Maas.

http://www.win.tue.nl/~bdeweger/downloads/MT%20Martijn%20Maas.pdf

[10] Short Programs for functions on Curves

https://crypto.stanford.edu/miller/miller.pdf

[11] FIPS PUB 186-4. Digital Signature Standard (DSS)

http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[12] Certicom ECC Tutorial

https://www.certicom.com/10-introduction

[13] Elliptic Curve Cryptography: ECDH and ECDSA

http://andrea.corbellini.name/2015/05/30/elliptic-curve-cryptography-ecdh-and-ecdsa/

[14] The Elliptic Curve Digital Signature Algorithm (ECDSA)

http://cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf

[15] Andreas Enge. Bilinear pairings on elliptic curves, 2013.

https://hal.archives-ouvertes.fr/file/index/docid/767404/filename/pairings.pdf

[16] Tom St. Denis. Cryptography for Developers 1st Edition

[17] Darrel Hankerson, Alfred J. Menezes, Scott Vanstone. Guide to Elliptic Curve Cryptography 2004th Edition.

[18] Miguel Ángel Rodríguez Roselló. 8088-8086-8087 PROGRAMACION ENSAMBLADOR EN ENTORNO MS-DOS

[19] Inline assembly for x86 in Linux

http://www.ibm.com/developerworks/library/l-ia/

[20] Atmel Inline Assembler Cookbook

http://www.atmel.com/webdoc/AVRLibcReferenceManual/inline_asm.html

9. APÉNDICE A. LIBRERÍA PBC (PAIRING BASED CRIPTOGRAPHY)

La librería PBC (Paring Based Cryptography) constituye sin lugar a dudas el estado del arte en lo que a implementaciones de este tipo de sistemas criptográficos se refiere. Es una librería desarrollada en C y está construida sobre la librería GMP (GNU Multiple Precission Arithmetic Library). PBC ha sido desarrollada por Ben Lync y cuenta con una excelente documentación, ejemplos y gran cantidad de comentarios en los ficheros fuente que hacen legible el código. Se trata además de una librería de código abierto sujeta a los términos de licencia LGPL (GNU Lesser General Public License), lo que la convierte en ideal para ser integrada en otros programas.

Dada la dependencia de la librería GMP, que a su vez depende fuertemente de otras librerías de sistema, no puede ser integrada dentro del kernel de iPXE. Sin embargo, los algoritmos escritos en C sí que pueden ser extraídos e integrados en cualquier otro programa escrito en C haciendo algunas modificaciones. De esta librería se ha obtenido el código base para el cálculo del emparejamiento de Tate, basado en el algoritmo de Miller.

Esta librería constituye un punto de partida perfecto para aquellos que desean trabajar con criptografía basada en emparejamientos bilineales y por ese motivo hemos considerado interesante detallar cómo se compila para su uso inmediato en un sistema Linux.

La web de los autores es la mejor referencia para encontrar documentación y guías de uso de la librería PBC: https://crypto.stanford.edu/pbc/manual/

Veamos los pasos necesarios para compilar la librería y un pequeño ejemplo de uso. En primer lugar deberemos descargar el código fuente desde el sitio oficial de PBC:

https://crypto.stanford.edu/pbc/download.html

Como se ha mencionado, esta librería está implementada sobre la librería GMP. Por tanto, necesitaremos también el código fuente de esta para poder satisfacer la dependencia. Lo obtendremos igualmente del sitio oficial de GMP:

https://gmplib.org/#DOWNLOAD

El siguiente paso es compilar ambas librerías. Primero GMP y a continuación PBC. La librería PBC nos obliga a referenciar la librería GMP en tiempo de compilación y por tanto es necesario compilar esta última en primer lugar.

Una vez descargado el código fuente de GMP, la compilación es muy sencilla. Basta con posicionarse en el directorio donde se han descargado los ficheros fuente y ejecutar las siguientes líneas:

```
]$./configure
]$ make
```

A continuación procederemos a compilar la librearía PBC pero antes de hacerlo es necesario referenciar la librería GMP recién compilada exportando las variables de entorno LDFLAGS y CPPFLAGS como sigue:

```
]$ export LDFLAGS='-L/home/tato/Teleco/PFC/GMPBigNum/gmp-6.0.0/.libs/'
]$ export CPPFLAGS='-I/home/tato/Teleco/PFC/GMPBigNum/gmp-6.0.0/'
```

donde /home/tato/Teleco/PFC/GMPBigNum/gmp-6.0.0/ es el directorio en el que se ha descargado el código fuente de GMP.

En este momento procedemos a compilar la librería PBC ejecutando las siguientes instrucciones:

```
./configure
make
make install
```

Si no se referencia correctamente la librería GMP, la compilación de la librería fallará, advirtiendo que falta la librería GMP y que es necesario añadirla a LDFLAGS.

Para comprobar que la librería ha sido correctamente compilada e instalada en el equipo seguiremos un pequeño ejemplo que, a su vez, ilustra el uso básico de los emparejamientos bilineales. Este ejemplo se puede encontrar en el sitio oficial de la librearía en el siguiente enlace:

https://crypto.stanford.edu/pbc/manual/ch01s03.html

El ejemplo consiste en la lanzar la aplicación de línea de comando pbc obtenida durante la compilación de la librería e ilustrar cómo se hace un uso de básico de los emparejamientos bilineales. La aplicación pbc es una calculadora de emparejamientos bilineales.

Para lanzar la aplicación PBC simplemente ejecutaremos el binario *pbc*. A continuación introduciremos la secuencia de comandos que se puede ver en la imagen:

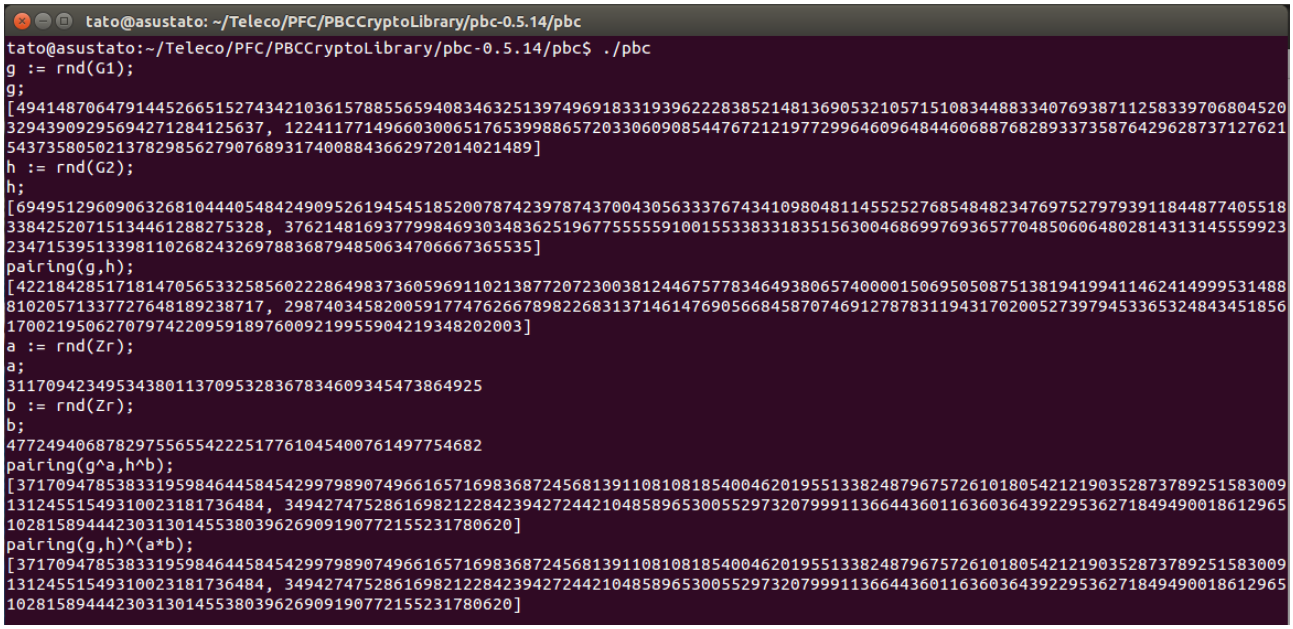


Ilustración 29 - Ejemplo de uso de la calculadora de emparejamientos bilineales de PBC

En este ejemplo se ilustra la propiedad de bilinealidad de los emparejamientos bilineales. Aunque no se indica de manera explícita, se está haciendo uso del emparejamiento de Weil, que, como ya hemos visto, es un emparejamiento simétrico.

En primer lugar se comienza por declarar dos puntos: h y g. El punto g pertenece al grupo G1, mientras que el punto h pertenece al grupo G2. Como el emparejamiento de Weil es simétrico, G1 y G2 son en realidad el mismo grupo, de orden r. A continuación se calcula el emparejamiento de h y g, obteniéndose como resultado un nuevo punto (de otro grupo distintos a G1 y G2) cuyas coordenadas x e y se imprimen por pantalla. No se hará nada con el punto obtenido. Solamente se trata de ilustrar cómo se calcula el emparejamiento de dos puntos.

A continuación se obtienen dos números aleatorios a y b cuyo valor están comprendidos entre 1 y r, dado que los grupos G1 y G2 (que son el mismo) son de orden r. Los números a y b pertenecen al campo finito subyacente a la curva (de la que no sabemos nada y ha sido elegida por defecto por la aplicación pbc). Con estos dos números aleatorios y los puntos h y g se calculan los siguientes valores: $e(g^a,h^b)$ y $e(g,h)^{ab}$, donde se ha sustituido la notación *pairing(g,h)* por *e(g,h)* por coherencia con el resto de la memoria.

Por la propiedad de bilinealidad de los emparejamientos el resultado de ambos cálculos debe ser idéntico.

10. APÉNDICE B. EJEMPLO DE SOPORTE A CURVAS ELIPTICAS EN APLICACIONES

En este anexo solamente pretendemos ilustrar la presencia de soporte a la tecnología de curvas elípticas en dos aplicaciones de las que todos somos usuarios. Estamos hablando de la Máquina Virtual de Java de Oracle (anteriormente Sun) y del navegador Google Chrome.

Comencemos por la Máquina Virtual de Java. La siguiente tabla ilustra cómo se va incluyendo soporte para algoritmos de curva elíptica desde la versión 1.4 hasta la versión 1.7. Lo que se puede deducir es que a partir de la versión 1.6 Oracle incluyó soporte para la criptografía basada en curvas elípticas.

JAVA J2SE CIPHER SUITE SUPPORT	v1.4	v1.4.2	v1.5	v1.6	v1.7
TLS_DH_anon_WITH_AES_256_CBC_SHA256					√
TLS_**ECDH**_anon_WITH_AES_256_CBC_SHA					√
TLS_DH_anon_WITH_AES_256_CBC_SHA		√	√	√	√
TLS_DH_anon_WITH_AES_128_CBC_SHA256					√
TLS_**ECDH**_anon_WITH_AES_128_CBC_SHA				√	√
TLS_DH_anon_WITH_AES_128_CBC_SHA		√	√	√	√
TLS_**ECDH**_anon_WITH_RC4_128_SHA				√	√
SSL_DH_anon_WITH_RC4_128_MD5	√	√	√	√	√
TLS_**ECDH**_anon_WITH_3DES_EDE_CBC_SHA				√	√
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	√	√	√	√	√
TLS_RSA_WITH_NULL_SHA256					√
TLS_**ECDHE_ECDSA**_WITH_NULL_SHA				√	√
TLS_**ECDHE**_RSA_WITH_NULL_SHA				√	√
SSL_RSA_WITH_NULL_SHA	√	√	√	√	√
TLS_**ECDH_ECDSA**_WITH_NULL_SHA				√	√
TLS_**ECDH**_RSA_WITH_NULL_SHA				√	√
TLS_**ECDH**_anon_WITH_NULL_SHA				√	√
SSL_RSA_WITH_NULL_MD5	√	√	√	√	√
SSL_RSA_WITH_DES_CBC_SHA	√	√	√	√	√
SSL_DHE_RSA_WITH_DES_CBC_SHA		√	√	√	
SSL_DHE_DSS_WITH_DES_CBC_SHA	√	√	√	√	
SSL_DH_anon_WITH_DES_CBC_SHA	√	√	√	√	
SSL_RSA_EXPORT_WITH_RC4_40_MD5	√	√	√	√	
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	√	√	√	√	
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA		√	√	√	
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA		√	√	√	
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	√	√	√	√	
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	√	√	√	√	
TLS_KRB5_WITH_RC4_128_SHA			√	√	
TLS_KRB5_WITH_RC4_128_MD5			√	√	
TLS_KRB5_WITH_3DES_EDE_CBC_SHA			√	√	
TLS_KRB5_WITH_3DES_EDE_CBC_MD5			√	√	
TLS_KRB5_WITH_DES_CBC_SHA			√	√	
TLS_KRB5_WITH_DES_CBC_MD5			√	√	
TLS_KRB5_EXPORT_WITH_RC4_40_SHA			√	√	
TLS_KRB5_EXPORT_WITH_RC4_40_MD5			√	√	
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA			√	√	
TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5			√	√	

Tabla 5 - Soporte a algoritmos de curvas elípticas de la Máquina Virtual de Java

Veamos ahora el caso del navegador Google Chrome. Esta vez no veremos la evolución entre versiones, sino únicamente el soporte criptográfico de una versión concreta (v49.0.2623.112). Se trata de una versión actual del conocido navegador. Lo más importante del listado es que aparece en orden de preferencia. Esta preferencia ha sido definida por el fabricante, Google, y, como se puede ver, en las primeras posiciones aparecen los algoritmos basados en curvas elípticas.

GOOGLE CHROME v49.0 CIPHER SUITE SUPPORT	ORDEN DE PREFERENCIA	KEYSIZE (bits)
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	1	128
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	2	128
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	3	256
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	4	256
OLD_TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	5	256
OLD_TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	6	256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	7	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	8	256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	9	128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	10	128
TLS_RSA_WITH_AES_128_GCM_SHA256	11	128
TLS_RSA_WITH_AES_256_CBC_SHA	12	256
TLS_RSA_WITH_AES_128_CBC_SHA	13	128

Tabla 6 - Soporte a algoritmos de curvas elípticas del navegador Google Chrome

11. APÉNDICE C. FORTALEZA Y VALIDEZ TEMPORAL DE LAS CLAVES

La siguiente tabla recoge una serie de recomendaciones del NIST en lo que al uso de claves para los distintos algoritmos simétricos criptográfico se refiere. La tabla establece relaciones de equivalencia para la longitud de las claves entre algoritmos de clave asimétricas, algoritmos de clave simétrica y curvas elípticas. También ofrece las fechas estimadas hasta las cuales se considera que dichas longitudes de claves serán seguras.

Algoritmo y Longitud de clave	Nivel de seguridad	Equivalencia EC	Validez temporal
RSA/DSA 1024 bits	80 bits	160 bits	Hasta 2010
RSA/DSA 2048 bits	112 bits	224 bits	2011 – 2030
RSA/DSA 3072 bits	128 bits	256 bits	Superior 2030
RSA/DSA 7680 bits	192 bits	384 bits	Muy superior a 2030
RSA/DSA 15360 bits	256 bits	512 bits	Muy superior a 2030

Tabla 7 - Fortaleza y validez temporal de las claves en función de su tamaño según el NIST

La primera columna (Algoritmo y Longitud de clave) describe el algoritmo asimétrico y la longitud de las claves empleadas.

La segunda columna (Nivel de seguridad) establece una equivalencia entre la dificultad para romper la seguridad del algoritmo con la longitud indicada con respecto a la dificultad de romper la seguridad en el caso de un algoritmo de clave simétrica. Por ejemplo, el NIST considera que el algoritmo RSA con claves de 1024 es tan bueno como un algoritmo simétrico que utiliza longitud de claves de 80 bits.

La tercera columna (Equivalencia EC) estable una relación de equivalencia aproximada entre la longitud de clave que debe utilizarse en conjunto con el algoritmo asimétrico y la longitud de clave que proporciona el mismo nivel de seguridad cuando se trabaja con curvas elípticas.

La última columna (Validez temporal) indica hasta qué fecha el NIST considera que las claves del tamaño indicado en la primera columna se consideran seguras y por tanto pueden ser utilizadas para construir cripto-sistemas.

12. PLIEGO DE CONDICIONES

El presente proyecto aborda el problema de la securización de las comunicaciones que tienen lugar en tiempo de pre-boot para un determinado tipo de sistemas terminales interconectados por una red de comunicaciones TCP/IP. Estos sistemas, que reciben el nombre de endpoints, presentan una serie de limitaciones relativas al hardware con el que están equipados.

La ejecución de este proyecto da lugar al diseño e implementación de una solución que va a permitir esta securización de las comunicaciones conforme a los requisitos establecidos.

12.1. ENTREGABLES

Memoria del Proyecto

Original y dos copias del Proyecto, encuadernadas de forma normalizada. Cesión a la Universidad Autónoma de Madrid, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, para que pueda ser utilizada de forma libre y gratuita por todos los usuarios del repositorio y del portal e-ciencia, los derechos de reproducción, de distribución, de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual.

Implementación del prototipo

La implementación del prototipo consta de:

- Librería criptográfica implementada en C y en ensamblador en línea. Se entregan ficheros fuente (.c) y de cabeceras (.h)
- Ficheros o scripts de configuración.
- Programas auxiliares y de pruebas.

12.2. CONDICIONES DE DESARROLLO

Recursos Hardware:

- Equipo portátil ASUS F550C con procesador Intel Core i7, 8 GB de RAM y disco duro SATA de 500 GB a 7.200 r.p.m. utilizado para el desarrollo del prototipo.
- Servidor HP Proliant DL 380p con procesador Intel® Xeon® E5-2609 v2 de 4 núcleos a 2.5GHz, 16 GB de RAM y 2 TB de disco SAS a 10.000 r.p.m. utilizado para albergar el servidor de control de versiones de código Apache Subversion y para ejecutar el servidor genérico y el Centro Generador de Claves en la fase de validación.
- Kiosco Talaris CashStar 9110 utilizado en la fase de validación del prototipo.
- Kiosco Wincor ProCash 1500 utilizado en la fase de validación del prototipo.

Recursos Software:

- Sistema Operativo Ubuntu LTS-12.0.4 64 bits utilizado en el equipo de desarrollo.
- Sistema Operativo CentOS 6.6 64 bits en el Servidor Proliant.
- Sistema Operativo Windows XP Service Pack 3 32 bits en kiosco Talaris CashStar.
- Sistema Operativo Windows 7 Professional 32 bits en kiosco Wincor ProCash.

- Servidor Apache Subversion v1.8 para Linux x64.
- Software hipervisor de virtualización Oracle Virtual Box v4.3 utilizado en el equipo de desarrollo para trabajar con máquinas virtuales.
- Microsoft Word 2010 para la redacción de la memoria.

13. PRESUPUESTO

A continuación desglosamos el presupuesto total invertido para la realización de proyecto. Identificamos las siguientes partidas:

- Presupuesto de ejecución material.
- Gastos generales y Beneficio industrial.
- Honorarios por la redacción y dirección del proyecto.
- Presupuesto total.

El presupuesto de ejecución material, junto a los gastos generales y el beneficio industrial forman parte del denominado presupuesto por ejecución por contrata que, junto a los honorarios percibidos por la redacción y dirección del proyecto, conforman el presupuesto de costes total. Todas estas cantidades están expresadas en euros.

13.1. PRESUPUESTO DE EJECUCIÓN MATERIAL

El Presupuesto de Ejecución Material (PEM) se compone de los costes de mano de obra y de los costes de los recursos materiales empleados para la realización del proyecto. La partida presupuestaria relativa a los honorarios por la dirección del proyecto es considerada al margen.

13.1.1. DESCOMPOSICIÓN EN TAREAS

El coste de la mano de obra se calcula a partir de las tareas y los perfiles de los trabajadores que las realizan. A continuación se enumeran la tareas en las que este proyecto se ha divido junto con su duración y el perfil profesional de la persona que la ha ejecutado.

Tarea 1: Estudio del problema y establecimiento de los requisitos del sistema

Objetivo: estudiar con detenimiento el problema y redactar los requisitos que debe cumplir la solución.

Duración: 0.50 meses.

Esfuerzo: 0.25 persona/mes. Ingeniero Superior.

Tarea 2: Estudio del estado del arte e investigación

Objetivo: estudiar el estado del arte de las comunicaciones en pre-boot, el estándar PXE, la Criptografía de Curva Elíptica y la Criptografía Basada en Identidad.

Duración: 4 meses.

Esfuerzo: 2 persona/mes. Ingeniero Superior.

Tarea 3: Diseño de la solución

Objetivo: Basándose en la investigación realizada, diseñar de manera conceptual la solución.

Duración: 4 meses.

Esfuerzo: 2 personas/mes. Ingeniero Superior.

Tarea 4: Implementación del prototipo

Objetivo: implementar un prototipo funcional del sistema basado en el diseño producido en la tarea anterior.

Duración: 8 meses.

Esfuerzo: 4 persona/mes. Ingeniero Superior.

Tarea 5: Validación del prototipo

Objetivo: diseñar y ejecutar pruebas que permiten validar el correcto funcionamiento de la solución e identificar problemas que deban ser solventados.

Duración: 2 meses.

Esfuerzo: 1 personas. Ingeniero Superior.

Tarea 6: Recodificación

Objetivo: subsanar los errores y deficiencias encontrados durante la fase de validación del prototipo.

Duración: 1 mes.

Esfuerzo: 0.5 personas/mes. Ingeniero Superior.

Tarea 7. Validación del prototipo

Objetivo: repetir las pruebas funcionales del sistema para asegurarse de que todos los errores han sido reparados correctamente y de que el prototipo queda libre de cualquier malfuncionamiento.

Duración: 1 mes.

Esfuerzo: 0.5 personas/mes. Ingeniero Superior.

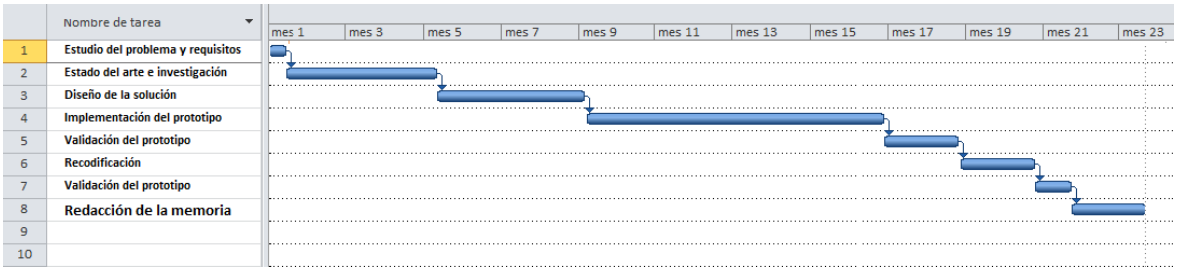
Tarea 8. Redacción de la memoria

Objetivo: documentar todo el estudio realizado en las fases iniciales, el trabajo llevado a cabo y exponer las conclusiones obtenidas en la finalización del proyecto, incluyendo potenciales tareas a realizar como trabajo futuro.

Duración: 2 meses.

Esfuerzo: 1 personas/mes Ingeniero Superior y 0.5 personas/mes Administrativo.

A continuación se ilustra en un diagrama de Gantt las tareas realizadas con sus respectivas duraciones e interdependencias. Como se puede observar, la ejecución completa del proyecto tiene una duración de 22.5 meses.



13.1.2. COSTES DE LOS RECURSOS HUMANOS O MANO DE OBRA

Para la realización del proyecto se requieren dos perfiles distintos de trabajador:

- Un Ingeniero Superior, que será el encargado de todas las tareas técnicas asociadas al proyecto: investigación, definición de requisitos, diseño, implementación y validación.
- Un Administrativo, que se hará cargo de la preparación y redacción de la documentación asociada al proyecto.

Para la estimación de los costes asociados a la mano de obra de los perfiles mencionados, se asume que la cotización está sujeta al Régimen General de la Seguridad Social para el ejercicio 16. La tabla que se expone a continuación recoge las bases de cotización mínima y máxima para los distintos grupos de cotización.

BASES DE COTIZACIÓN RÉGIMEN GENERAL EJERCICIO 16 - ORDEN ESS/70/2016, de 29/01 (BOE del 30/01)		
Grupos de Cotización	**Base mínima**	**Base máxima**
	Euros (Mes)	Euros (Mes)
1	1.067,40	3.642,00
2	885,30	3.642,00
3	770,10	3.642,00
4	764,40	3.642,00
5	764,40	3.642,00
6	764,40	3.642,00
7	764,40	3.642,00
	Euros (Día)	Euros (Día)
8	25,48	121,40
9	25,48	121,40
10	25,48	121,40
11	25,48	121,40
TOPE Máximo	3.642,00	
TOPE Mínimo	764,40	

Tabla 8 - Bases de cotización por grupos para el ejercicio 16

Se asume que ambos perfiles de trabajador realizan una jornada laboral de 8 horas diarias y que un mes consta de 21 días hábiles. Por otra parte, y coincidiendo con el valor máximo establecido por el XVII Convenio Colectivo Nacional Empresas de Ingeniería y Oficinas (Documento BOE-A-2013-11199) al que están sujetos los contratos de ambos perfiles de trabajador, el número de horas de trabajo efectivo a lo largo del ejercicio completo será de 1.800 horas.

El Ingeniero superior pertenece al grupo de cotización 1, mientras que el Administrativo pertenece al grupo 7.

Con arreglo a estos datos, se calculan los costes salariales asociados a los trabajadores. El resultado es el siguiente:

COSTES SALARIALES		
CONCEPTO	**INGENIERO SUPERIOR**	**ADMINISTRATIVO**
Base de cotización máxima	43.704,00 €	43.704,00 €
Contingencias comunes (23.6%)	7.434,00 €	4.413,20 €
Desempleo, F.G.S y Formación profesional (7.5%)	2.362,50 €	1.402,50 €
Coste de la seguridad social	9.796,50 €	5.815,70 €
Salario bruto anual	31.500,00 €	18.700,00 €
Coste salarial anual	41.296,50 €	24.515,70 €
Coste salarial por hora	22,94 €	13,62 €
Número de horas	1.890,00 h	84 h
Coste total	43.361,32 €	1.144,08 €

Tabla 9 – Desglose de los costes salariales

Por lo que el coste total de la mano de obra es el que se expone en la siguiente tabla:

COSTE TOTAL MANO DE OBRA	
CONCEPTO	**VALOR**
Ingeniero Superior de Telecomunicaciones	43.361,32 €
Administrativo	1.144,08 €
Coste total	**44.505,40 €**

Tabla 10 - Coste total de la mano de obra

13.1.3. COSTE DE LOS RECURSOS MATERIALES

Para la realización de proyecto es imprescindible disponer de una serie de recursos materiales software y hardware. La adquisición de todos los recursos se amortizará en un período de tres años. A continuación se detalla el coste de todos estos recursos:

COSTE TOTAL DE LOS RECURSOS MATERIALES	
CONCEPTO	**VALOR**
Recursos Hardware	
Equipo portátil de desarrollo ASUS F550C	699,00 €
Ampliación memoria RAM 4 GB	54,95 €
Servidor HP Proliant DL 380p	2,315,00 €
Kiosco Talaris CashStar 9110 (reacondicionado)	2.250,00 €
Kiosco Wincor ProCash 1500 (reacondicionado)	2.970,00 €
Recursos Software	
Licencia Windows 7 Professional 32 bits	139,00 €
Licencia Windows XP Professional 32 bits	69,00 €
Licencia Microsoft Office 2010	109,00 €
Coste total	**8.605,95 €**

Tabla 11 - Coste de los recursos materiales

13.1.4. COSTE TOTAL DE LOS RECURSOS

La suma del coste de la mano de obra con el coste de los recursos materiales da lugar al Presupuesto de Ejecución Material. En consecuencia, el desglose final del PEM es el siguiente:

PRESUPUESTO DE EJECUCIÓN MATERIAL	
CONCEPTO	**VALOR**
Coste mano de obra	44.505,40 €
Coste recursos materiales	8.605,95 €
Coste total	**53.111,35 €**

Tabla 12 - Presupuesto de ejecución material

13.2. GASTOS GENERALES Y BENEFICIO INDUSTRIAL

Los gastos generables engloban todas aquellas partidas que derivan de la utilización de instalaciones, cargas fiduciarias, amortizaciones, gastos fiscales, etc. Atendiendo a esta partida de gastos y teniendo en cuenta el beneficio industrial establecido por la empresa, el Presupuesto de Ejecución por Contrata queda desglosado como sigue:

PRESUPUESTO DE EJECUCIÓN POR CONTRATA	
CONCEPTO	**VALOR**
Presupuesto de ejecución material	53.111,35 €
Gastos generales (15% del PEM)	7.966,70 €
Beneficio industrial (6% del PEM)	3.186,68 €
Coste total	**64.264,73 €**

Tabla 13 - Presupuesto de ejecución por contrata

13.3. HONORARIOS POR REDACCIÓN Y DIRECCIÓN DEL PROYECTO

Los honorarios por la redacción y los honorarios por la dirección del proyecto quedan establecidos en el 5% del presupuesto de ejecución por contrata.

HONORARIOS POR REDACCIÓN Y DIRECCIÓN DEL PROYECTO	
CONCEPTO	**VALOR**
Honorarios por dirección del proyecto	5.006,08 €
Honorarios por redacción de la documentación	5.006,08 €
Coste total	**10.012,16 €**

Tabla 14 - Honorarios por dirección y redacción

13.4. PRESUPUESTO TOTAL

Sumando todas las partidas de gastos anteriormente detalladas y aplicando el 21% adicional correspondiente al Impuesto sobre el Valor Añadido (IVA) se obtiene el presupuesto final del proyecto.

PRESUPUESTO TOTAL	
CONCEPTO	**VALOR**
Presupuesto de ejecución por contrata	64.264,73 €
Honorarios profesionales	10.012,16 €
Impuesto sobre el Valor Añadido (21%)	15.598,14 €
Coste final	**89.875,03 €**

Tabla 15 - Presupuesto total

El presupuesto final del proyecto asciende a la cantidad de OCHENTA Y NUEVE MIL OCHOCIENTOS SETENTA Y CINCO euros con TRES céntimos.

Madrid, Junio de 2016.

Ángel Francisco Zato del Corral

El ingeniero proyectista.