



Trabajo Final de grado

Creación de una LiveCD de instalación personalizada basada en SecureBoot

Autor: Igor González Lerma

Tutor: Joaquín López Sánchez-Montañes

A mi mujer Saioa y mi hija Eider por las horas robadas y su
apoyo incondicional.

A Endika amigo, compañero y mentor sin
el que nada de todo esto hubiese
sido posible.

A mis profesores del Grado Superior.

A todos, ¡muchas gracias!

Contexto y justificación del Trabajo

Dada la creciente necesidad de seguridad derivada de la implantación de los planes Industria 4.0 en el mundo industrial, existen una serie de normas que requieren validar y asegurar la integridad del software que se ejecuta sobre un dispositivo. Para ello, en algunos casos se puede presentar la necesidad de acceder a dispositivos y herramientas de seguridad criptográfica como los integrados en *UEFI* (especialmente *SecureBoot*) y módulos *TPM*. En conjunción con esto, la integridad queda validada mediante imágenes de solo lectura tipo *Squashfs*.

Actualmente no existen métodos rápidos y directos para arrancar imágenes *SecureBoot Ready* (con certificados personalizados) capaces de instalar imágenes cifradas que se apoyen en el *TPM* para el descifrado propio del disco.

La conjunción de estos requisitos de seguridad, con una imagen de instalación *LiveCD*, da lugar a un sistema de rápido despliegue de equipos certificados mediante normas como LINCE (España) o *Common Criteria* (Nivel Mundial).

Palabras clave: LiveCD, SecureBoot, TPM, Certificados, Imágenes cifradas.

Object

Given the growing need for security derived from the implementation of Industry 4.0 plans in the industrial world, there are a number of standards that require the validation and integrity of the software running on a device. For this reason, in some cases there may be a need to access cryptographic security devices and tools such as those integrated in *UEFI* (specially *SecureBoot*) and *TPM* modules. In addition, the integrity is validated by *Squashfs*-type read-only images.

Nowadays, there are no fast and direct methods for booting *SecureBoot Ready* images (with custom certificates) which can be installed encrypted images which use *TPM* to decrypt the disk itself.

The combination of these security requirements with a *LiveCD* installation image results in a system of rapid deployment of equipment certified by standards such as LINCE (Spain) or *Common Criteria* (World Level).

Keywords: LiveCD, SecureBoot, TPM, certificates, encrypted images.

Índice

Tabla de contenido

Contexto y justificación del Trabajo.....	3
Object.....	3
Índice.....	4
1.- Introducción.....	6
1.1.- Contexto y justificación del trabajo.....	6
1.3.- <i>Hardware</i> de trabajo.....	7
Las características.....	7
La descripción física.....	7
Dimensiones y peso.....	8
1.2.- Objetivos del trabajo.....	9
Por ello se debe cumplir los siguientes puntos.....	9
2.- Calendario del proyecto.....	10
2.1.- Planificación temporal.....	10
2.2.- Diagrama de Gantt.....	12
2.3.- Recursos necesarios.....	13
2.4.- Productos obtenidos.....	14
2.5.- Breve descripción de los otros capítulos de la memoria.....	15
3.- Estado del arte.....	16
3.1.- Conclusiones obtenidas.....	17
4.- Tecnologías usadas.....	18
4.1.- Secure BOOT.....	18
Certificado x509.....	18
4.2.- LVM.....	21
4.3.- LUKS.....	22
4.4.- TPM.....	23
Almacenamiento Non-Volatile.....	23
Endorsement Key.....	24
Attestation Identity Keys.....	24
Platform Configuration Register.....	24
Random Number Generator.....	24
SHA.....	24
RSA.....	25
Justificación del uso de <i>TPM</i> en este proyecto.....	25
4.5.- Initramfs.....	25
El proceso de arranque de linux.....	25
4.6.- SquashFS.....	27
Características del sistema de archivos.....	27
Diseño del sistema de archivos.....	28
4.7.- OverlayFS.....	29
Upper y Lower.....	29
4.8.- Principales Herramientas.....	31

4.8.1.- Chroot.....	31
4.8.2.- Squashfs-tools.....	31
4.8.3.- OpenSSL.....	32
4.8.4.- Sbsigntool.....	32
4.8.5.- Xorriso.....	32
4.8.6.- EfiBOOTmgr.....	33
4.8.7.- dd.....	33
4.8.8.- Checkinstall.....	33
4.8.9.- Objcopy.....	34
5.- Implementando la solución.....	35
5.1.- Configuración Hardware.....	35
5.2.- Fichero Pressed.....	37
5.2.1.- Fichero randompass.sh.....	39
5.2.2.- Fichero <i>efiBOOT.sh</i>	39
5.2.3.- Fichero script <i>_ch.sh</i>	40
5.2.4.- Pasos para la compilación de paquetes .deb del <i>tpm</i>	40
5.3.- Creación de los certificados.....	43
5.4.- Creación del <i>BOOT</i> firmado y con herramientas del <i>TPM</i>	44
5.5.- <i>Initrd</i>	47
5.5.- Imagen base.....	48
5.6.- Imagen ISO arrancable desde un live-CD firmada.....	50
5.7.- Instalación.....	52
5.8.- Pruebas realizadas.....	55
5.8.1.- Comprobar el sistema.....	55
5.8.2.- Desactivar <i>SecureBoot</i>	57
5.8.3.- <i>USB</i> sin firmar.....	58
6.- Conclusiones y continuidad del trabajo realizado.....	59
6.1.- Próximos pasos.....	60
7.- Glosario.....	61
8.- Índice de figuras.....	65
9.- Bibliografía.....	67

1.- Introducción

1.1.- Contexto y justificación del trabajo

La Industria 4.0 combina técnicas avanzadas de producción y operaciones con tecnologías inteligentes con el objetivo de alcanzar *smart factories* con una mayor automatización, interconexión y globalización. Dichos avances permiten optimizar los procesos de fabricación, su mantenimiento así como la integración con otros sistemas y procesos.

Esto genera la necesidad de disponer de sistemas que operen a través de redes privadas y públicas, compartiendo información y cooperando con el objetivo de obtener una mejora en la productividad. Con el aumento de la conectividad y el uso de protocolos industriales, surge la necesidad de proteger los sistemas industriales mediante comunicaciones seguras, así como de la integridad del software que actualmente se está ejecutando en dichos dispositivos.

Para poder cumplir estos requisitos de integridad del software dentro de un producto, surge la obligación de acceder a dispositivos y herramientas de seguridad criptográfica como *TPM(Trusted Platform Module)* y *SecureBoot* sobre *UEFI*. Del mismo modo, el producto se debe apoyar en imágenes de solo lectura de tipo *Squashfs* sobre particiones cifradas para evitar la alteración del software base.

[*Engimedia*](#), empresa ubicada en San Sebastian y donde actualmente me encuentro como empleado, construye productos nativos de ciberseguridad ICS/OT que permiten a los clientes fabricantes y a las infraestructuras críticas proteger sus activos estratégicos, mejorar la productividad evitando los tiempos de inactividad, mejorar la seguridad de las personas y cumplir con las normas y recomendaciones internacionales.

En la actualidad *Engimedia* no dispone de ningún método automatizado que permita la configuración del sistema con dichos requisitos de seguridad: permitir el arranque de imágenes *SecureBoot Ready* con certificados personalizados almacenados en la BIOS; además descifrado de disco mediante una clave almacenada en el *TPM* junto con una política de acceso mediante *PCRs (Platform Configuration Register)*.

Con este objetivo surge la necesidad de este proyecto, buscando crear una imagen de instalación *LiveCD* que permita configurar el equipo con los requisitos de seguridad mencionados, a la vez de los propios requisitos del producto suministrados por *Engimedia* (particiones del disco duro y *hardware* de trabajo "Dell edge gateway 3003").

1.3.- Hardware de trabajo

La elección como *hardware* del modelo “[Dell edge gateway 3003](#)” se debe a que se trata de uno de los productos disponibles dentro del catálogo de *Enigma* y porque cumple los requisitos necesarios para poder afrontar el proyecto.

Las características

- Procesador Intel® Atom™ E3815 a 1,46 GHz (512 kB de caché de nivel 2).
- Ubuntu Server 18.04.
- 2 GB de memoria DDR3L a 1067 MHz.
- Almacenamiento eMMC de 32GB.
- TPM.

La descripción física

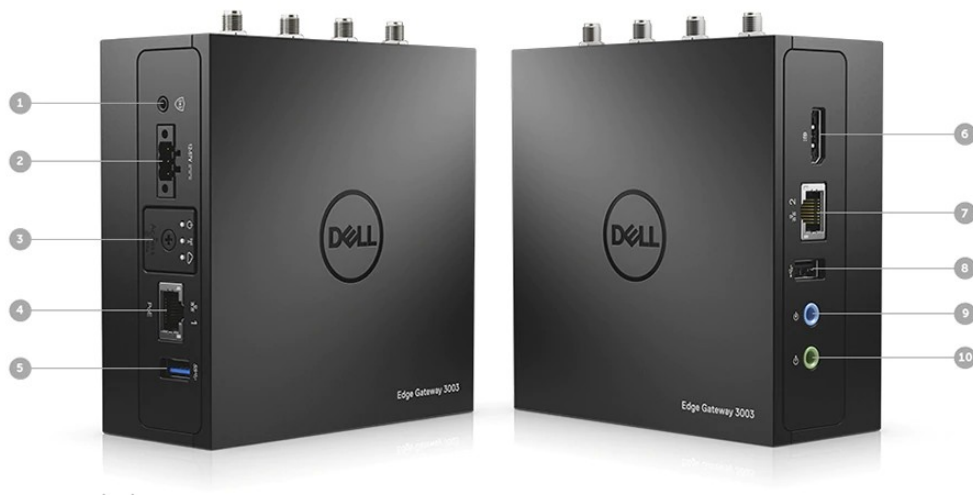


Fig 1: Dell descripción física

1. Entrada del interruptor de intrusión externo
2. Conector de alimentación/entrada de CC con patilla de encendido/reactivación
3. LED de estado y ranuras para tarjetas (micro-SIM y micro-SD)
4. Fast Ethernet (RJ-45) con PoE
5. Puerto USB 3.0
6. DisplayPort1.1
7. Fast Ethernet (RJ-45) con PoE
8. USB 2.0

9. Audio: entrada de línea de 3,5 mm

10. Audio: salida de línea de 3,5mm

Dimensiones y peso



Fig 2: Dell Dimensiones y Peso

1. Altura: 125mm (4,9")

2. Anchura: 125mm (4,9")

3. Profundidad: 51mm (2")

Peso mínimo: 1 kg (2,2libras)

1.2.- Objetivos del trabajo

El proyecto tiene como objetivo obtener una imagen de *LiveCD* basada en Linux que permita la instalación y configuración del modo más desatendido posible de un sistema base, sobre el *hardware Dell edge gateway 3003*. Cumpliendo los estándares de seguridad que aseguren la integridad del sistema, junto con las necesidades establecidas por *Enigmedia*.

Por ello se debe cumplir los siguientes puntos.

- *LiveCD* basada en Linux.
- *Hardware* configurado con *SecureBoot* y Certificados personalizados.
- Modelo de *hardware* sobre el que realizar las pruebas y la obtención del producto *Dell edge gateway 3003*.
- Instalador con particionado automático de particiones *LVM* cifradas con *LUKS* mediante clave *aleatoria*.
- Almacenamiento de clave aleatoria en el *TPM* con política de acceso mediante *PCRs*.
- Sistema Base mediante *Squashfs* (sistema de archivos comprimidos de solo lectura) y *overlayfs*.
- *BOOT* firmado y con herramientas automáticas de descifrado de disco.

2.- Calendario del proyecto

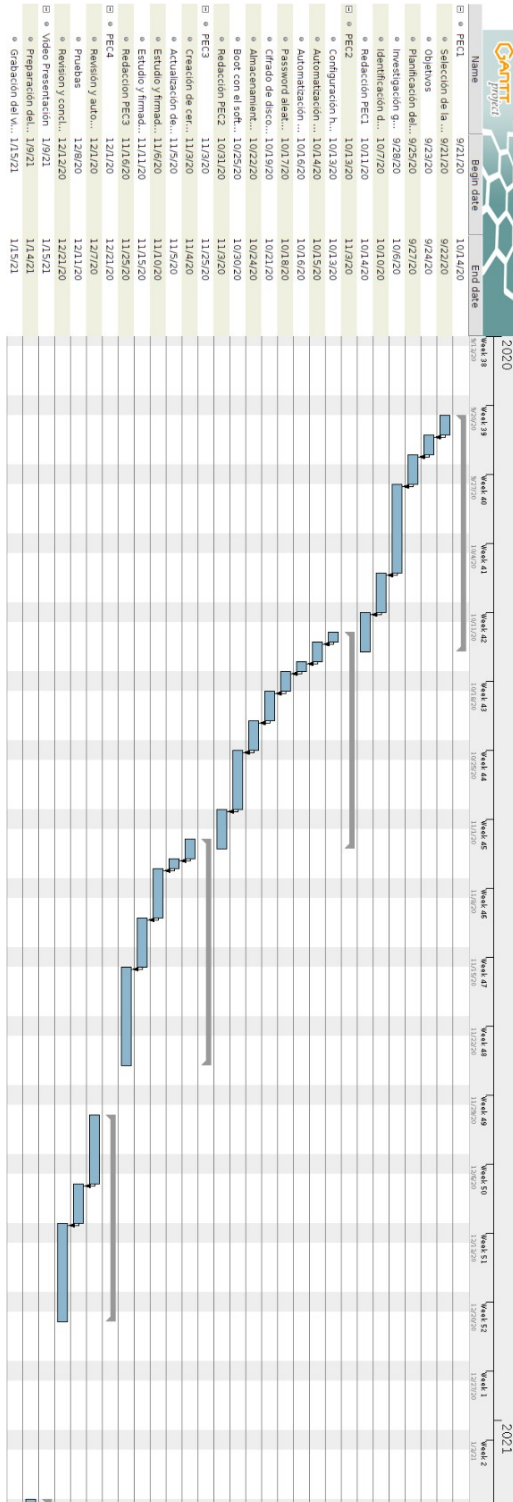
Con el objetivo de tener todo el proyecto finalizado en las fechas indicadas se prevé utilizar el método de evaluación continua propuesto por la UOC, así como de las indicaciones establecidas por el consultor.

2.1.- Planificación temporal

Nombre de la tarea	Horas dedicadas
PEC1 Plan de trabajo (21/09/2020 - 12/10/2020)	92
Selección de la temática y justificación.	5
Objetivos.	5
Planificación del trabajo.	12
Investigación general.	40
Identificación de los puntos conflictivos .	20
Redacción PEC1.	10
PEC2 (13/10/2020 - 02/11/2020)	66
Configuración <i>hardware</i> .	1
Automatización formateo de las particiones.	10
Automatización configuración base.	2
<i>Password</i> aleatoria y presentación en pantalla.	5
Cifrado de disco con <i>password</i> aleatoria.	10
Almacenamiento <i>password</i> en <i>TPM</i> .	10
<i>BOOT</i> con el software necesario para obtener <i>password</i> del <i>TPM</i> y descifrado del disco.	18
Redacción PEC2.	10
PEC3 (03/11/2020 - 30/11/2020)	50
Creación de certificados personalizados.	5
Actualización de la configuración de <i>hardware</i> .	5
Estudio y firmado <i>BOOT LiveCD</i> .	10
Estudio y firmado <i>BOOT Instalada</i> .	10
Redacción PEC3.	20
PEC4 (01/12/2020 - 21/12/2020)	60
Revisión y automatización de procesos.	20
Pruebas.	10
Revisión y conclusión de la memoria.	30
Video Presentación (09/01/2021 - 15/01/2021)	17

Preparación del material	15
Grabación del video	2
TOTAL	285

2.2.- Diagrama de Gantt



2.3.- Recursos necesarios

Para el desarrollo del proyecto se utilizarán los siguientes equipos y herramientas:

Hardware:

- PC portátil con instalación *ubuntu 18.04*.
- *Dell edge gateway 3003*.

Software:

- Debian *installer*.
- Paquetes *apt*:
 - *efitools 1.4.2+git20140118-0ubuntu2*.
 - *sbsigntool 0.6-3.2ubuntu2*.
 - *Xorriso 1.4.8-3*.
 - *openssl1.0*.
 - *syslinux-utils*.
 - *Squashfs-tools*.
 - *chrootuid*
 - *Checkinstall*
- Paquetes deb:
 - *libsapi0_2.0-1_amd64.deb*.
 - *tpm2_1.0-1_amd64.deb*.
 - *tpm2-tools_3.0.2-1_amd64.deb*.

2.4.- Productos obtenidos

Los productos obtenidos a la finalización del proyecto.

- *LiveCD*.
- Software de automatización para la creación de *LiveCDs*.
- Memoria con las fases del proyecto y la descripción de los objetivos del trabajo.
- Presentación multimedia.

2.5.- Breve descripción de los otros capítulos de la memoria

En este primeros capítulos se han introducido los principales objetivos del trabajo. A la vez, haciendo uso de técnicas propias de la gestión de proyectos, se ha planificado su desarrollo.

En los siguientes capítulos de la memoria se exponen una breve descripción de las tecnologías usadas, asimismo de los pasos desarrollados para obtener la *ISO* final. Se presenta una breve descripción a continuación:

- **Capítulo 3. Estado del arte:** Se analiza la situación actual de las diferentes opciones que permiten obtener *ISOs* personalizadas.
- **Capítulo 4. Tecnologías usadas:** Se presentan brevemente las tecnologías usadas: *SecureBoot*, *LVM*, *LUKS*, *TPM* y *PCRs*, *Squashfs*, *Overlayfs*, *Initramfs*, y las principales herramientas usadas.
- **Capítulo 5. Implementando la solución.** Información de la implementación del producto y resultado de las pruebas realizadas.
- **Capítulo 6. Conclusiones y continuidad del trabajo realizado:** Conclusiones del producto obtenido, posibles mejoras implementables al sistema y necesidades de cara a su uso.
- **Capítulo 7. Glosarios:** Conjunto de palabras empleadas en la memoria y que puedan resultar de difícil comprensión.
- **Capítulo 8. Índice de Figuras.** Breve descripción de las imágenes presentadas.
- **Capítulo 9. Bibliografía:** Referencia a los documentos y páginas web consultadas para la realización del proyecto.

3.- Estado del arte

En el capítulo actual se presentan algunas de las aplicaciones que permiten obtener ciertos de los requisitos de seguridad establecidos como objetivos. Se investigan y se analizan las características y funcionalidades con el objetivo de poder personalizarlas y adaptarlas al producto buscado durante el desarrollo del siguiente TFG.

- *Cubic o Custom Ubuntu ISO Creator*: se trata de una interfaz gráfica que posibilita crear una *ISO* personalizada. Aunque permite una gran personalización, solo es operativa para distribuciones Debian. Además, no se localiza la opción de firmar, durante la creación de la *ISO*, el *BOOT* con los certificados personalizados ni la forma de interactuar con el *TPM* durante la instalación.

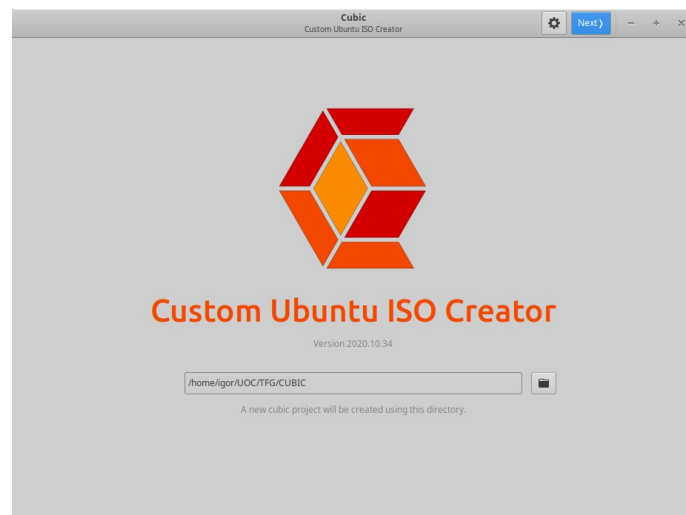


Fig 3: Cubic

- *Systemback*, permite realizar copias de seguridad del sistema, restauración del sistema y creación de un archivo *ISO* de arranque. Este sistema construye una *ISO* personalizada a partir de un sistema ya instalada. En este caso como en el anterior no se localiza la opción de firmar durante la creación de la *ISO* el *BOOT* y tampoco la integración con el *TPM* durante la instalación.

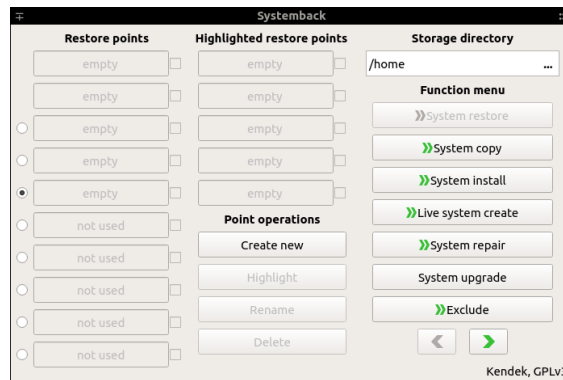


Fig 4: Systemback

3.1.- Conclusiones obtenidas

Actualmente no se halla ninguna herramienta que permita la creación de la *ISO* cumpliendo todas las necesidades establecidas:

- *Secure BOOT* o arranque seguro.
- Formateo y cifrado de particiones mediante *LUKS*.
- Interacción con *TPM* y *PCRs*
- *Initramfs*
- *Squashfs* y *overlays*

Con el análisis de requisitos expuesto se realiza una búsqueda de software de calidad suficiente para cumplir los objetivos. Se localizan manuales y herramientas que permiten cumplir los requisitos de seguridad establecidos para el producto individualmente.

Dichas herramientas y manuales son libres y se basan en el sistema operativo Linux, siendo quizá la distribución basada en Debian la de mayor fuente de información.

La decisión de basarse en el instalador de red de Debian como base del proyecto se debe a la alta personalización que permite y al mínimo de paquetes que instala.

4.- Tecnologías usadas

4.1.- Secure BOOT

SecureBoot o arranque seguro: se trata de un modo para *UEFI* que impide la ejecución de software no firmado o certificado durante el arranque. *SecureBoot* usa técnicas criptográficas de *checksum* y firmas, cada programa que es cargado por el *firmware* debe incluir una firma y un *checksum* de verificación para validar que el programa es confiable. Esto evita que código inesperado / no autorizado se ejecuta en el entorno *UEFI*.

La especificación de *UEFI* define:

- Un formato para almacenar certificados X.509 en variables *UEFI*.
- Un método para validar el *BootLoader* y los *drivers*
- Un mecanismo para la revocación de certificados y firmas.

Certificado x509

Para que un clave pública se pueda considerar válida y asociada a un usuario determinado debe tener un certificado que así lo demuestre. Un certificado digital es una estructura de datos que contiene información del propietario de las claves criptográficas, la clave pública en sí y una firma digital de los campos anteriores que le da validez.

X509 es un formato estándar para certificados de clave pública firmados por una autoridad certificadora (CA):

En un certificado hay tres tipos principales de campos:

- Básicos: campos que aportan información sobre la autoridad de certificación que ha emitido el certificado (CA), la entidad/suscriptor a que pertenece la clave pública, la propia clave pública y el periodo de validez y la identificación del certificado.
- Necesarios para la firma: campos que utilizará quien reciba el certificado para comprobar que el documento ha sido firmado correctamente.

- Ampliaciones: campos que han aparecido para cubrir las nuevas necesidades de atributos en un certificado.

Listas de revocación (CRL):

Los certificados cuentan con un periodo de validez que puede ir de unos meses a unos años. Durante ese periodo de tiempo que el certificado es válido la entidad certificadora que lo generó mantiene información sobre el estado de ese certificado. Una lista de revocación de certificados (*CRL*) es una estructura de datos firmada por la autoridad de certificación emisora, que contiene los datos necesarios para determinar el estado de un certificado.

La principal información que guarda es el “*estado de anulación*”, que indica que el periodo de validez del certificado ha finalizado antes de tiempo y el sistema que lo emplee no debe confiar en él.

Tipos de UEFI Keys:

- *Platform Key (PK)*: Suministrada por el vendedor *hardware*. Permite la manipulación de la *KEK*.
- *Key Exchange Key (KEK)*: El *KEK* permite la manipulación de la *db* y la *dbx*.
- *Authorized DB (db)*: Identifica el código de confianza.
- *Unathorized DB (dbx)*: Identifica código comprometido o código malicioso.

Además, la especificación describe dos modos *SecureBoot*. El primero es el modo de configuración, permite guardar o eliminar certificados y *hashes* en el almacenamiento *UEFI*. Este sistema permite configurar el *SecureBoot*. Y modo usuario, en el que se restringe la manipulación de los certificados almacenados.

La siguiente imagen muestra cómo funciona el *SecureBoot*. Si el *BootLader* no puede garantizar que sea confiable no arrancará.

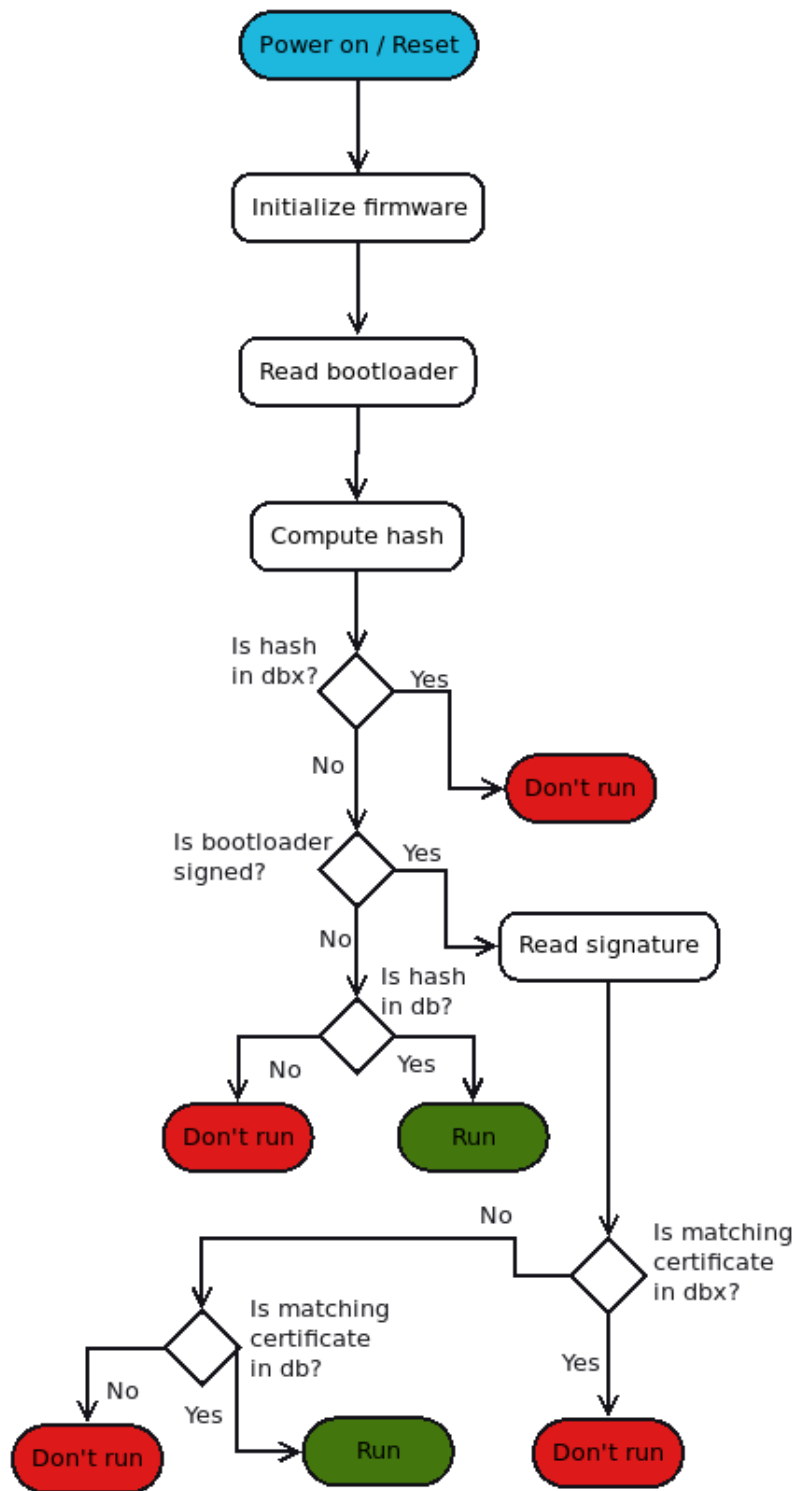


Fig 5: SecureBoot

4.2.- LVM

LVM es una utilidad que permite crear volúmenes lógicos a partir de discos duros físicos, es decir, se trata de una capa de abstracción entre un dispositivo de almacenamiento y un sistema de ficheros haciendo uso de la función *device-mapper* del *kernel* de Linux para proporcionar un sistema de particiones independientes de la estructura subyacente del disco.

Los bloques básicos que construyen *LVM* son:

- *Physical volume*: Nodo de dispositivo de bloque de *Unix*, utilizable para almacenamiento por *LVM*. Representa el nivel más bajo de un *LVM* y alberga un encabezado *LVM*.
- *Volume group*: Grupo de volúmenes físicos que sirve de depósito para volúmenes lógicos, es decir, una colección de uno o más Volúmenes Físicos.
- *Logical volume*: los grupos de volumen se dividen en un volumen lógico o varios volúmenes lógicos. Los volúmenes lógicos son dispositivos de bloque de Unix análogos a las particiones físicas.
- *Physical extent*: Para manipular los verdaderos datos, se divide en bloques de datos llamados Extensiones Físicas. La extensión contigua más pequeña (por defecto 4 MiB) en el volumen físico que se puede asignar a un volumen lógico.
- *Logical Extents*: Análogas a las *Physical extent*, pero a nivel de Volumen Lógico. El tamaño de los bloques es el mismo para cada Volumen Lógico (LV) del mismo Grupo de Volúmenes (VG).

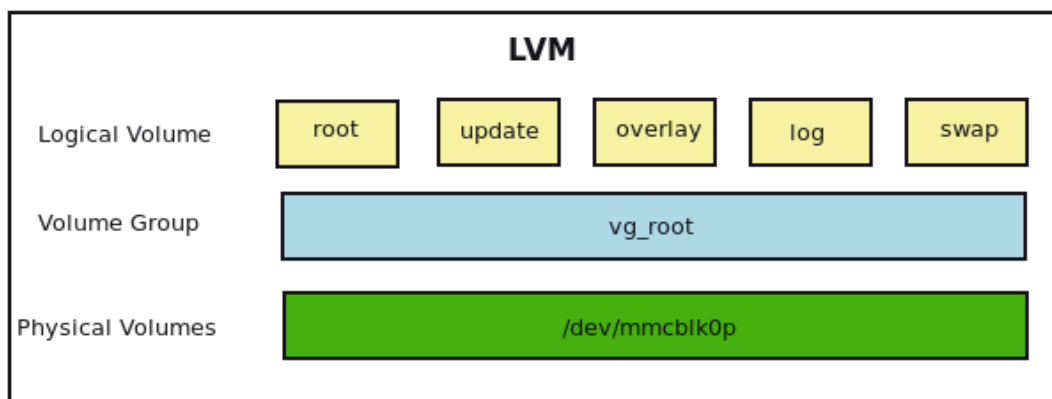


Fig 6: LVM de la práctica

4.3.- LUKS

Linux Unified Key Setup se trata de una especificación de cifrado de disco creado por *Clemens Fruhwirth*, principalmente para Linux. *LUKS* especifica un formato estándar en disco, independientemente de la plataforma, facilitando la compatibilidad y su uso entre diferentes programas y garantizando la gestión de contraseñas en un lugar seguro.

El diseño de *LUKS* puede ser usado con cualquier cifrado o modo de cifrado, pero por razones de compatibilidad, *LUKS* estandariza los nombres cifrados y los modos de cifrado. A continuación, se expone un listado de los tipos de cifrado, el modo de cifrado y los tipos de hash:

Cifrados simétricos:

- *AES.*
- *Serpent.*
- *Twofish.*

Modos:

- *CDC-Plain.*
- *CBC-Plain64.*
- *CBC-ESSIV.*
- *XTS-Plain.*
- *XTS-Plain64.*

Hashes:

- *PBKDF2-HMAC-SHA1.*
- *PBKDF2-HMAC-SHA256.*
- *PBKDF2-HMAC-SHA512.*
- *PBKDF2-HMAC-RipeMD160.*

La implementación de referencia funciona en Linux y se basa en una versión mejorada de *cryptsetup*, utilizando *dm-crypt* como la interfaz de cifrado de disco.

Dm-crypt es un subsistema de cifrado de discos que provee cifrado transparente de dispositivos de bloque utilizando la utilidad *cryptoapi* del kernel de Linux 2.6. Puede cifrar discos completos, particiones, volúmenes *RAID* por software, volúmenes lógicos y archivos. Se muestra como un dispositivo de bloque, que se puede utilizar para respaldar sistemas de archivos, espacios de intercambio o como un volumen físico *LVM*.

4.4.-TPM

TPMs son dispositivos que están adheridos a las especificaciones de la plataforma *Trusted Computing Group's*. Están diseñados para la realización de tareas criptográficas pero no son aceleradores criptográficos.

Los bloques que contiene un *TPM* son los siguientes:

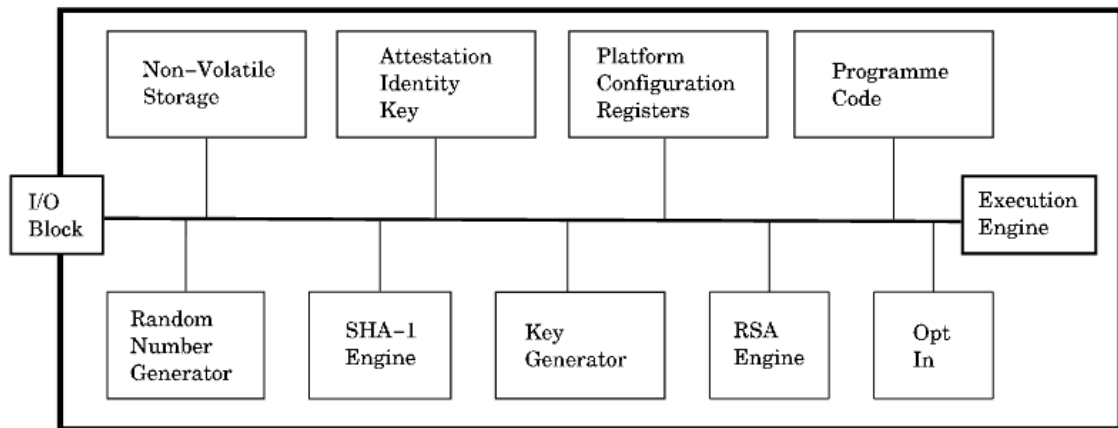


Fig 7:

TPM

Almacenamiento Non-Volatile:

Los *TPM* cuentan con una memoria *non-volatile* para el almacenamiento de claves. En ella se encuentra el Endorsement Key(EK) y la *Storage Root Key(SRK)*.

El *TPM* también usa la memoria *non-volatile* para almacenar los datos de autorización. Dichos datos de autorización (contraseña del propietario) se establecen durante la toma de propiedad del *TPM*.

Endorsement Key

Se trata de una pareja de claves única, la parte privada no es accesible y es única por cada *TPM*. Mientras que la parte pública se trata de un certificado para un uso limitado de operaciones, como la toma de propiedad del *TPM* o la generación de clave *AIK*.

Attestation Identity Keys

Esta clave se puede ver como un alias de la *EK*. Cada *TPM* puede tener varias *AIK*, para mantener el anonimato entre varios servicios que requieren la prueba de identidad. Los *AIK* por lo tanto deben almacenarse de forma segura, aunque se pueden mantener en el *TPM* se recomienda almacenarlos en un almacenamiento externo seguro.

Platform Configuration Register

Los *PCRs* son usados para mantener la integridad del sistema y es una característica para comprobar que recursos estratégicos del sistema no han sido alterados. Se puede asociar la ejecución o no de un código en función del estado en que se encuentren determinados *PCRs*. No obstante, esta práctica resulta peligrosa debido a que actualizaciones de software o *firmware* alteran el *hash* de dichos *PCRs*.

Random Number Generator

Una característica de los *TPM* es la inclusión de un verdadero generador de bits aleatorios, el cual puede usarse para la construcción de claves para cifrados simétricos o la generación de *nonce*.

SHA

Motor de generación de *HASH* dada una entrada.

RSA

Motor de generación de claves *RSA*. Debido a que la generación de claves es computacional compleja, el *TPM* cuenta con un engine dedicado a *RSA* el cual se puede usar para firmar, encriptar y desencriptar.

Justificación del uso de *TPM* en este proyecto

El uso del *TPM*, junto con la política definida sobre los *PCRs*, garantizan que el disco duro no puede ser descifrado salvo en las condiciones explícitamente definidas. De esta forma la clave de descifrado se aloja en un chip físico gestionado por el *TPM* y accesible solamente cuando el arranque se encuentra firmado con nuestros propios certificados y la tecnología *SecureBoot* está activada.

4.5.-Initramfs

Se trata de un sistema de archivos *RAM* de inicio, fundamentados en *tmpfs*, con el objetivo de montar el sistema de archivos raíz mínimo para permitir la ejecución de programas antes de que se monte el verdadero sistema de archivos raíz. No emplea un dispositivo de bloque separado tal y como hace *initrd* y por tanto no se ejecuta ningún tipo de almacenamiento en caché.

Todos los ficheros, bibliotecas, herramientas, ajustes de configuración, es decir: el conjunto completo de directorios se ubican dentro de un único archivo *cpio*. Dicho archivo se comprime mediante el algoritmo de compresión *gzip* y se almacena junto al núcleo de Linux.

El proceso de arranque de linux

El proceso de arranque de Linux consta de varias etapas. En la siguiente lista se resume brevemente el proceso de arranque:

- *UEFI*. Después de encender la computadora, *UEFI* inicializa la pantalla y el teclado y prueba la memoria principal. Cuando se reconoce el primer disco duro,

y se verifican las firmas del fichero de arranque, el control del sistema pasa de *UEFI* al *BootLoader*.

- *BootLoader*. El primer sector de datos físico de 512 *bytes* del primer disco duro se carga en la memoria principal y el cargador de arranque que reside al principio de este sector toma el control. Los primeros 512 bytes del primer disco duro se denominan también *Master BOOT Record* (MBR). El cargador de arranque pasa, en este caso, al núcleo de Linux.

- *Kernel e initramfs*. El gestor de arranque carga tanto el *kernel* como un sistema de ficheros inicial basado en *RAM* (*initramfs*) en la memoria. El *initramfs* contiene el ejecutable *init* que se encarga de montar el verdadero sistema de archivos raíz.

- *init en initramfs*. *Init* realiza todas las acciones necesarias para montar el sistema de archivos raíz y proporcionar al núcleo el sistema de archivos necesario así como los controladores de dispositivos.

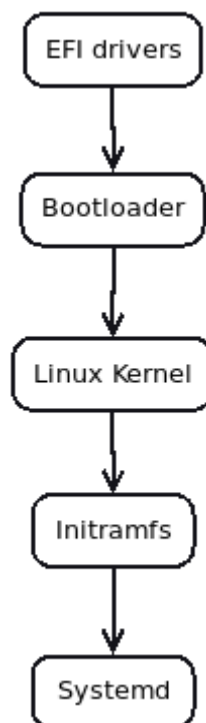


Fig 8: Proceso de arranque de linux

4.6.-SquashFS

Sistema de archivos comprimidos de solo lectura para Linux, que utiliza la compresión *zlib*, *lz4*, *lzo* o *xz* para comprimir archivos, inodos y directorios. Los inodos del sistema son muy pequeños y todos los bloques están empaquetados para minimizar la sobrecarga de datos. Los tamaños de bloque mayores de 4K son soportados hasta un máximo de 1Mbytes (tamaño de bloque por defecto 128K).

Características del sistema de archivos

- Tamaño máximo del sistema de archivos: 2^{64}
- Tamaño máximo del archivo: ~ 2 TiB
- Archivos máximos: ilimitados
- Máximo de directorios: ilimitado
- Máximo de entradas por directorio: ilimitado
- Tamaño máximo del bloque: 1 MiB
- Compresión de metadatos: sí
- Índices de directorio: sí
- Tiempo de creación del archivo: sí

Diseño del sistema de archivos

Un sistema de archivos *squashfs* consiste en un máximo de nueve partes, empaquetadas juntas en una alineación de bytes:



Fig 9: Squashfs

Los bloques de datos comprimidos se escriben en el sistema de archivos a medida que los archivos se leen desde el origen, y se comprueba si hay duplicados. Una vez que todos los datos del archivo han sido escritos, se rellenan las tablas completas de *inode*, directorio, fragmento, exportación, búsqueda de uid/gid y xattr.

4.7.-OverlayFS

Permite tener un árbol de directorios, normalmente de lectura-escritura, el cual es colocado sobre otro árbol de directorios de lectura-escritura. Todas las modificaciones van a una capa superior de escritura.

El enfoque del sistema de archivos *overlay* es "híbrido", porque los objetos que aparecen en el sistema de archivos no siempre parecen pertenecer a este sistema de archivos. En muchos casos, un objeto al que se accede en la unión será indistinguible del objeto correspondiente del sistema de archivos original.

Upper y Lower

Un sistema de archivos *overlay* combina dos sistemas de archivos: un sistema de archivos "superior" o "*Upper*" y un sistema de archivos "inferior" o "*Lower*". Cuando existe un nombre en ambos sistemas de archivos, el objeto en el sistema de archivos "*Upper*" es visible mientras que el objeto en el sistema de archivos "*Lower*" está oculto o, en el caso de los directorios, fusionado con el objeto "*Upper*".

Sería más correcto referirse a un "árbol de directorios" *Upper* y *Lower* que en lugar de a un "sistema de archivos", ya que es muy probable que ambos árboles de directorios estén en el mismo sistema de archivos y no es necesario que se indique la raíz de un sistema de archivos ni para el *Upper* ni para el *Lower*.

El sistema de archivos *Lower* puede ser cualquier sistema de archivos soportado por Linux y no necesita ser de escritura. El sistema de archivos *Upper* normalmente debe ser de escritura, soportar la creación de atributos extendidos `trusted.*`, y debe proporcionar un `d_type` válido en las respuestas `readdir`.

Ejemplo:

En la imagen presentada se presentan dos directorios, cada uno de los cuales contiene archivos y carpetas. El directorio "*Lower*" es de solo lectura. El acceso a los archivos a través del OverlayFS recupera primero los datos del directorio "*Upper*", y luego pasa al directorio "*Lower*" si un archivo no existe.

Las modificaciones de los archivos del directorio "*Upper*" se harán como de costumbre. Cualquier modificación de un archivo de la carpeta "*Lower*" creará una copia en el directorio "*Upper*", y ese archivo será el modificado. Esto deja los archivos base sin tocar y disponibles a través de un acceso directo a la carpeta "inferior".

Un archivo eliminado del directorio *OverlayFS* eliminaría directamente un archivo del directorio "*Upper*" y simularía esa eliminación del directorio "*Lower*" creando lo que se denomina un archivo "*whiteout*". Este archivo sólo existe dentro del directorio *OverlayFS*, sin aparecer físicamente en los directorios "*Upper*" o "*Lower*".

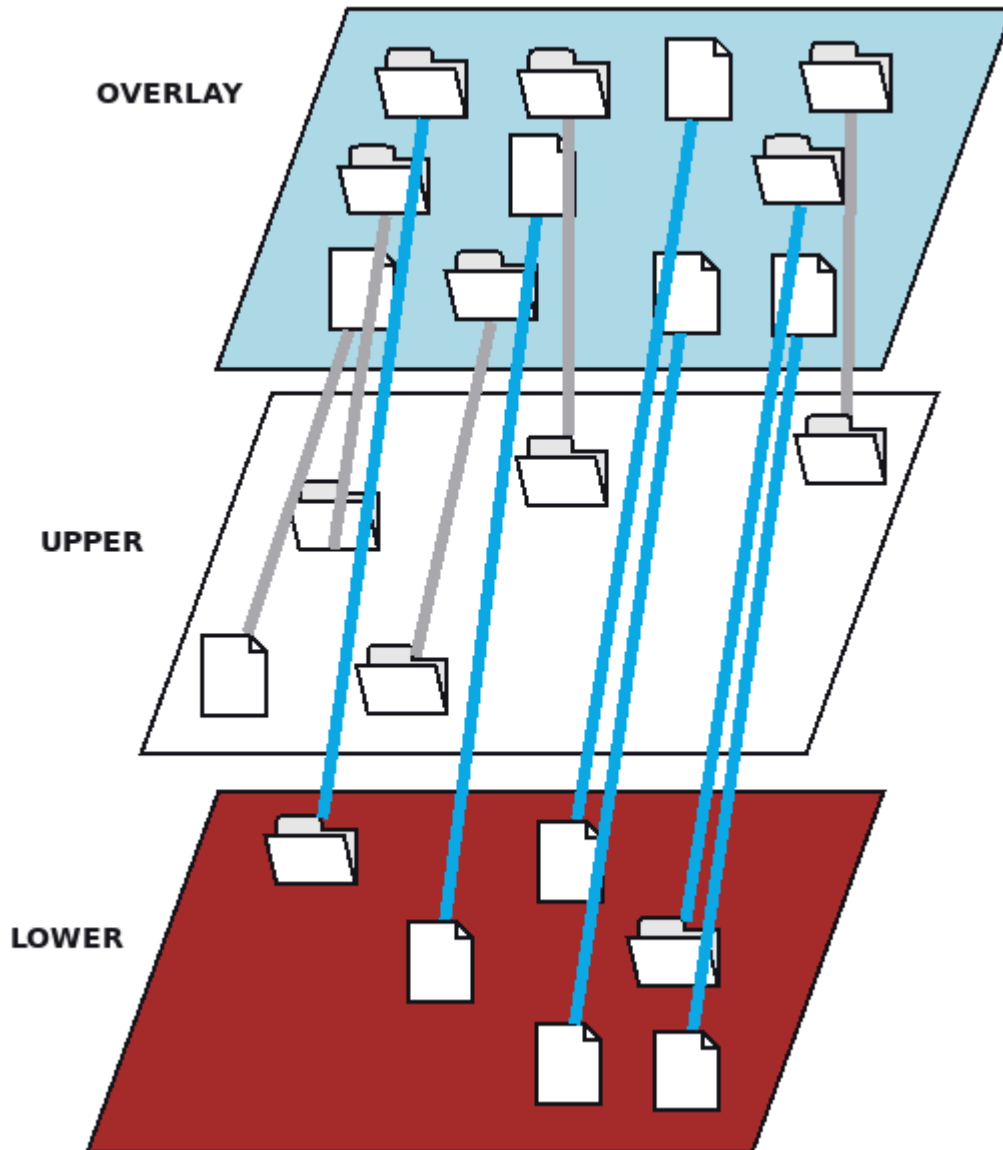


Fig 10: OverlayFS

4.8.- Principales Herramientas

4.8.1.- Chroot

Herramienta de Linux por la que se cambia el directorio *root* del disco (y el actual proceso y sus hijos) a otro directorio *root* simulado. Este nuevo entorno se conoce también como “jaula *chroot*”. Es importante destacar que desde la “jaula *chroot*” un usuario no podría acceder a los ficheros de fuera del entorno.

```
~ chroot --version
chroot (GNU coreutils) 8.28
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Roland McGrath.
```

Fig 11: Versión chroot

4.8.2.- Squashfs-tools

Herramienta de linux que permite gestionar imágenes *squashfs*, que se tratan de sistema de archivos de sólo lectura altamente comprimido para Linux.

Durante la práctica se hace uso de *squashfs* para comprimir la imagen y *unsquashfs* para su descompresión.

```
~ mksquashfs -version
mksquashfs version 4.3-git (2014/06/09)
copyright (C) 2014 Phillip Lougher <phillip@squashfs.org.uk>

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2,
or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
~ unsquashfs -version
unsquashfs version 4.3 (2014/05/12)
copyright (C) 2014 Phillip Lougher <phillip@squashfs.org.uk>

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2,
or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

Fig 12: Versión mksquashfs y unsquashfs

4.8.3.- OpenSSL

OpenSSL se trata de una aplicación con herramientas criptograficas multiusos. Durante el desarrollo de la practica su uso ha sido para la creación de los certificados.

```
~ openssl version
OpenSSL 1.1.1 11 Sep 2018
```

Fig 13: Versión OpenSSL

4.8.4.- Sbsigntool

Sbsigntool se trata de un paquete con utilidades de firma para el arranque seguro. Permite manipular firmas en binarios y *drivers* de *UEFI*. Para el desarrollo de la practica se hace uso de *sbsign* para firmar los *BOOT* y *sbverify* para verificarlos.

```
~ sbsign --version
sbsign 0.6
~ sbverify --version
sbverify 0.6
```

Fig 14: Versión sbsign y sbverify

4.8.5.- Xorriso

Xorriso herramienta de *linux* que crea, carga, manipula y escribe imágenes del sistema de archivos *ISO 9660* con extensiones *Rock Ridge*. Con dicha herramienta se crea el *Live-CD*.

```
~ xorriso --version
xorriso 1.4.8 : RockRidge filesystem manipulator, libburnia project.

xorriso 1.4.8
ISO 9660 Rock Ridge filesystem manipulator and CD/DVD/BD burn program
Copyright (C) 2017, Thomas Schmitt <scdbackup@gmx.net>, libburnia project.
xorriso version      : 1.4.8
Version timestamp   : 2017.09.12.143001
Build timestamp     : -none-given-
libisofs  in use    : 1.4.8 (min. 1.4.8)
libjte    in use    : 1.0.0 (min. 1.0.0)
libburn   in use    : 1.4.8 (min. 1.4.8)
libburn OS adapter: internal GNU/Linux SG_IO adapter sg-linux
libisoburn in use   : 1.4.8 (min. 1.4.8)
Provided under GNU GPL version 3 or later, due to libreadline license.
There is NO WARRANTY, to the extent permitted by law.
```

Fig 15: Versión xorriso

4.8.6.- EfiBOOTmgr

EfiBOOTmgr se trata de una herramienta que interactúa con el firmware *EFI* del sistema. Permite crear, editar, reordenar y eliminar entradas de arranque. Se hace uso de ella durante la instalación para poder introducir nuestro *BOOT* firmado.

```
~ efibootmgr --version
version 15
```

Fig 16: Versión efiBOOTmgr

4.8.7.- dd

dd es una utilidad de línea de comandos para Unix y sistemas operativos similares a Unix, cuyo propósito principal es convertir y copiar archivos. Utilizado para la copia de la imagen base al finalizar la instalación .

```
~ dd --version
dd (coreutils) 8.28
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Paul Rubin, David MacKenzie, and Stuart Kemp.
```

Fig 17: Versión dd

4.8.8.- Checkinstall

Checkinstall es un programa de *Unix* que permite la instalación de software compilado desde el código fuente para poder ser administrado por un gestor de paquetes. Durante el transcurso de la practica se hace uso de ella para la construcción de los paquetes *.deb* de las herramientas del *TPM* descargadas.

```
~ checkinstall -v
checkinstall 1.6.2, Copyright 2009 Felipe Eduardo Sanchez Diaz Duran
This software is released under the GNU GPL.

Copyright (c) 2009 Felipe Eduardo Sanchez Diaz Duran <izto@asic-linux.com.mx>

This program is free software; you can redistribute it and/or modify
it under the terms of the version 2 of the GNU General Public License
as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Fig 18: Versión checlinstall

4.8.9.- Objcopy

La utilidad GNU objcopy copia el contenido de un archivo objeto a otro. Puede escribir el archivo objeto de destino en un formato diferente al del archivo objeto de origen.

```
~ objcopy -V
GNU objcopy (GNU Binutils for Ubuntu) 2.30
Copyright (C) 2018 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.
```

Fig 19: Versión objcopy

5.- Implementando la solución

5.1.- Configuración Hardware

Para la configuración *hardware* se debe arrancar el equipo y presionar F2. En el menú de la BIOS en la parte izquierda seleccionar “Expert Key Management”:

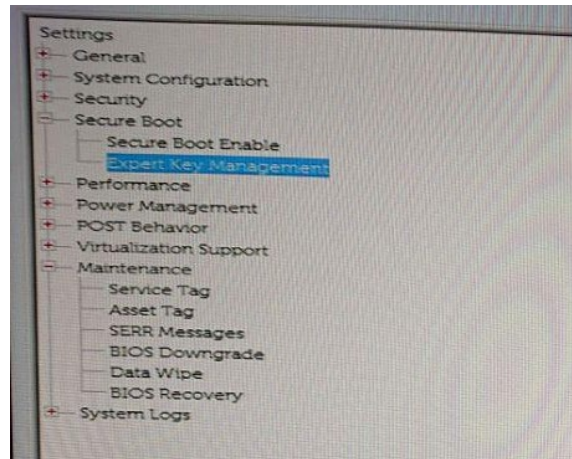


Fig 20: BIOS selección Expert Key Management

Una vez seleccionado “Expert Key Management”, en la parte derecha de la pantalla seleccionar “Enable Custom Mode”:

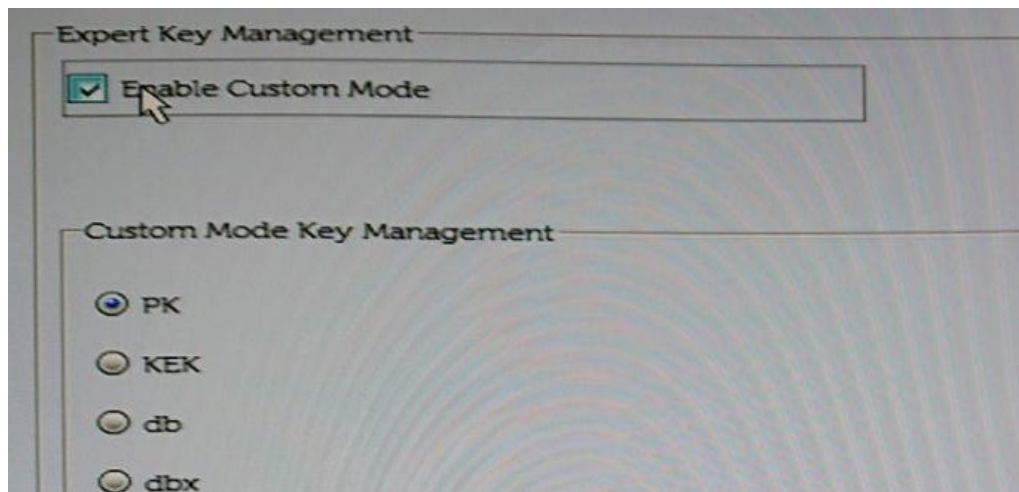


Fig 21: BIOS selección Enable Custom Mode

En la parte derecha seleccionar dentro de “Custom Mode Key Management” el “PK” y dar a “Replace from file”. En el navegador seleccionar el correspondiente PK.path:

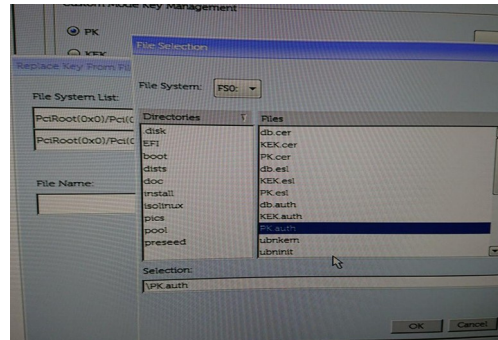


Fig 22: BIOS selección PK.auth

Repetir el proceso con “KEK” y con “db”:

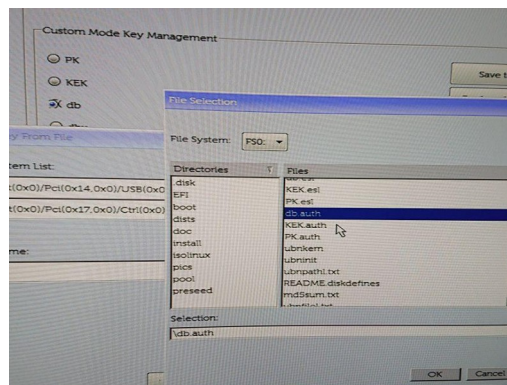


Fig 23: BIOS selección db.auth

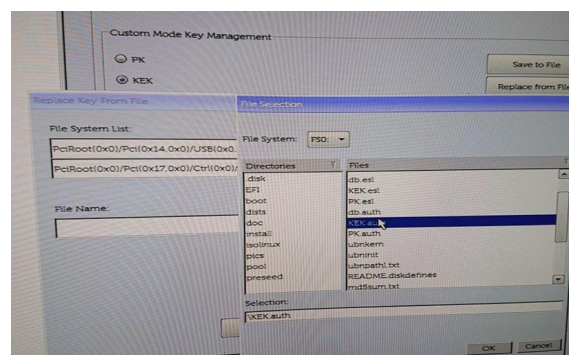


Fig 24: BIOS selección KEK.auth

5.2.- Fichero Preseed

El instalador de *Debian* ofrece la posibilidad, a través de ficheros de preconfiguración, de automatizar la instalación. El fichero *preseed* proporciona una manera de establecer las respuestas a las preguntas formuladas durante el proceso, sin tener que introducir manualmente las respuestas, mientras la instalación se está ejecutando. Esto hace posible automatizar completamente la mayoría de los tipos de instalación e incluso ofrece algunas características que no están disponibles durante las instalaciones normales.

El fichero *preseed* utilizado durante el desarrollo de la práctica se encuentra en el anexo 1. Cabe destacar dentro del fichero *preseed* algunas secciones que se detallan a continuación:

```
d-i pkgselect/exclude string tpm2 tpm2-tools libaspi0_2
```

Especifica que durante la instalación no se carguen los paquetes *TPM* de apt debido a que es necesario las herramientas del *TPM* en una versión más actual.

```
d-i partman/early_command string sh randompass.sh
```

El objetivo de este parámetro es lanzar un *script* durante la instalación, antes de la ejecución de *partman*. Posteriormente se detalla el objetivo de dicho *script*.

```
d-i partman-auto/*
```

Se encarga de definir el formateo de las particiones de forma desatendida para el usuario. En la figura se puede observar el objetivo final:

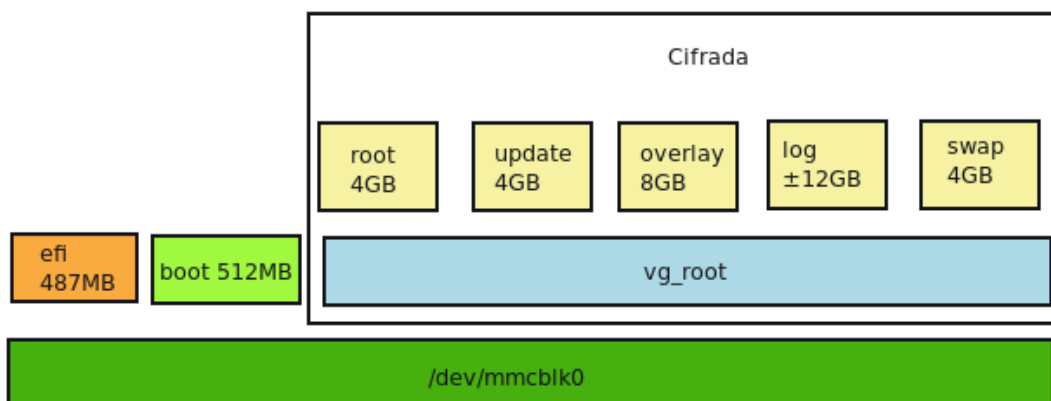


Fig 25: Particionado durante la instalación

d-i preseed/late_command string sh efiBOOT.sh

Con la configuración del siguiente parámetro, el *preseed* nos brinda la posibilidad de, finalizando la instalación, lanzar un *script*. El *script* definido en el anexo 3.

Una vez se tenga el equipo instalado, el particionado final quedara de la siguiente manera:

```
Disco /dev/mmcblk0: 28,5 GiB, 30601641984 bytes, 59768832 sectores
```

```
Unidades: sectores de 1 * 512 = 512 bytes
```

```
Tamaño de sector (lógico/físico): 512 bytes / 512 bytes
```

```
Tamaño de E/S (mínimo/óptimo): 512 bytes / 512 bytes
```

```
Tipo de etiqueta de disco: gpt
```

```
Identificador del disco: A56CD909-31B2-4BF8-BC7B-ABB35D27882F
```

```
Dispositivo Comienzo Final Sectores Tamaño Tipo
```

```
/dev/mmcblk0p1 2048 976895 974848 476M Sistema EFI
```

```
/dev/mmcblk0p2 976896 2344959 1368064 668M Sistema de ficheros de Linux
```

```
/dev/mmcblk0p3 2344960 59766783 57421824 27,4G Sistema de ficheros de Linux
```

```
Disco /dev/mapper/mmcblk0p3_crypt: 27,4 GiB, 29397876736 bytes, 57417728 sectores
```

```
Unidades: sectores de 1 * 512 = 512 bytes
```

```
Tamaño de sector (lógico/físico): 512 bytes / 512 bytes
```

```
Tamaño de E/S (mínimo/óptimo): 512 bytes / 512 bytes
```

```
Disco /dev/mapper/vg_root-swap: 1,9 GiB, 1996488704 bytes, 3899392 sectores
```

```
Unidades: sectores de 1 * 512 = 512 bytes
```

```
Tamaño de sector (lógico/físico): 512 bytes / 512 bytes
```

```
Tamaño de E/S (mínimo/óptimo): 512 bytes / 512 bytes
```

```
Disco /dev/mapper/vg_root-root: 3,7 GiB, 3997171712 bytes, 7806976 sectores
```

```
Unidades: sectores de 1 * 512 = 512 bytes
```

```
Tamaño de sector (lógico/físico): 512 bytes / 512 bytes
```

```
Tamaño de E/S (mínimo/óptimo): 512 bytes / 512 bytes
```

```
Disco /dev/mapper/vg_root-update: 3,7 GiB, 3997171712 bytes, 7806976 sectores
```

```
Unidades: sectores de 1 * 512 = 512 bytes
```

```
Tamaño de sector (lógico/físico): 512 bytes / 512 bytes
```

```
Tamaño de E/S (mínimo/óptimo): 512 bytes / 512 bytes
```

```
Disco /dev/mapper/vg_root-overlay: 7,5 GiB, 7998537728 bytes, 15622144 sectores
```

```
Unidades: sectores de 1 * 512 = 512 bytes
Tamaño de sector (lógico/físico): 512 bytes / 512 bytes
Tamaño de E/S (mínimo/óptimo): 512 bytes / 512 bytes
```

```
Disco /dev/mapper/vg_root-log: 10,6 GiB, 11404312576 bytes, 22274048 sectores
Unidades: sectores de 1 * 512 = 512 bytes
Tamaño de sector (lógico/físico): 512 bytes / 512 bytes
Tamaño de E/S (mínimo/óptimo): 512 bytes / 512 bytes
```

5.2.1.- Fichero `randompass.sh`

Script encargado de generar una *password random* durante la instalación y capturarlo en la variable `$PASS`. Además, permite la creación de un fichero temporal en `/tpm/variable.txt` para almacenar dicha variable y el tipo de disco detectado. También, presenta un cuadro de texto para la presentación del *password* por motivos de seguridad. El *script* se puede encontrar en el anexo 2.

5.2.2.- Fichero `efiBOOT.sh`

El *script* definido en el anexo 3 se encarga de realizar la copia de los ficheros necesarios:

- `BOOTX64.EFI` con las herramientas del *TPM* y firmado.
- `libsapi0_2.0-1_amd64.deb`, `tpm2-tools_3.0.2-1_amd64.deb`, `tpm2_1.0-1_amd64.deb` paquetes necesarios para interactuar con el *TPM*.

Asimismo, monta el *chroot* donde ejecuta el fichero `script_ch.sh` y al finalizar realiza la copia mediante el comando `dd` del sistema base de *Enigmedia systemimage.sqsh* al `vg_root-root`.

5.2.3.- Fichero script *ch.sh*

Script ejecutado dentro del *chroot* al finalizar la instalación, se localiza en el anexo 4 y sus principales acciones son:

- Se encarga de leer del fichero temporal creado en la ejecución del “*script ramdompass.sh*” tanto la *password* a guardar como el disco de la máquina.
- Instala los paquetes “.deb” necesarios para interactuar con el *TPM*.
- Almacena la *password* aleatoria dentro del *TPM*.
- Añade la entrada de “Enigmedia” con el *BOOT* firmado y modificado en el *firmware UEFI* del sistema . En caso que ya se encuentre una entrada anterior de “Enigmedia” se borra.

5.2.4.- Pasos para la compilación de paquetes .deb del *tpm*

1.- Descargar el código de los repositorios y seguir los pasos de compilación:

```
git clone https://github.com/tpm2-software/tpm2-tss.git  
git clone https://github.com/tpm2-software/tpm2-abrmd.git  
git clone https://github.com/tpm2-software/tpm2-tools.git
```

2.- Compilar *tpm2-tss*:

```
./BOOTstrap  
./configure --with-udevrulesdir=/etc/udev/rules.d  
make  
sudo udevadm control --reload-rules && sudo udevadm trigger  
sudo ldconfig  
sudo checkinstall
```

Rellenar con los siguientes datos:

```
Package: libsapi0  
Priority: extra  
Section: universe/libs  
Installed-Size: 9052  
Maintainer: info@Enigmedia.es  
Architecture: amd64  
Version: 2.0-1  
Depends: libc6 (>= 2.15)
```


Provides: libsapi0

Description: TPM2 Software stack library - TSS and TCTI libraries
 TPM2.0 TSS (Software Stack) consists of API layers provided to support
 TPM 2.0 chips. It is made out of three layers:

- .
- System API (SAPI), which implements the system layer API;
- TPM Command Transmission Interface (TCTI), which is used by SAPI to allow communication with the TAB/RM layer;
- Trusted Access Broker/Resource Manager (TAB/RM), which handles TPM resources and process coordination.
- .

This package contains the TSS and TCTI libraries that client applications will link against when t

3.- Compilar *tpm2-abrmd*:

```
sudo useradd --system --user-group tss
./BOOTstrap
./configure --with-dbuspolicydir=/etc/dbus-1/system.d --with-
systemdsystemunitdir=/lib/systemd/system
make
sudo checkinstall
```

Rellenar con los siguientes datos:

```
Package: tpm2
Priority: extra
Section: universe/tools
Installed-Size: 2820
Maintainer: info@Enigmaedia.es
Architecture: amd64
Version: 1.0-1
Depends: libsapi0 (>=2.0)
Provides: tpm2
Description: TPM2 ABRMD. Tpm Access Broker and resource manager utility.
```

4.- Compilar *tpm2-tools*:

```
./BOOTstrap
./configure
make
sudo checkinstall
```

Rellenar con los siguientes datos:

```
Package: tpm2-tools
Priority: extra
Section: universe/utils
```

```
Installed-Size: 5320
Maintainer: info@Enigmedia.es
Architecture: amd64
Version: 3.0.2-1
Depends: libc6 (>= 2.14), libcurl3-gnutls (>= 7.16.2), libsapi0 (>=
2.0), libssl1.1 (>= 1.1.0)
Provides: tpm2-tools
Description: TPM 2.0 utilities
 This package contains a set of tools to use with TPM 2.0 chips,
 for common tasks and features provided by the hardware; such as
 for doing basic key management, attestation, encryption and si
```

5.3.- Creación de los certificados

Dentro de la carpeta *certificates* existe un script(anexo 5) que genera los certificados X509 con la herramienta *openssl*:

```
#!/bin/bash
```

Se crea un identificador único:

```
uuidgen --random > GUID.txt
```

Se genera el *PK.auth* con la herramienta *openssl* y como common name "my Project":

```
openssl req -newkey rsa:2048 -nodes -keyout PK.key -new -x509 -sha256 -days 36500 -subj "/CN=my Project Key/" -out PK.crt
```

```
openssl x509 -outform DER -in PK.crt -out PK.cer
```

```
cert-to-efi-sig-list -g "$(< GUID.txt)" PK.crt PK.esl
```

```
sign-efi-sig-list -g "$(< GUID.txt)" -k PK.key -c PK.crt PK PK.esl PK.auth
```

Se genera el *KEK.auth* nuevamente con la herramienta *openssl* y como common name "my Project":

```
openssl req -newkey rsa:2048 -nodes -keyout KEK.key -new -x509 -sha256 -days 36500 -subj "/CN=my Key Exchange Key/" -out KEK.crt
```

```
openssl x509 -outform DER -in KEK.crt -out KEK.cer
```

```
cert-to-efi-sig-list -g "$(< GUID.txt)" KEK.crt KEK.esl
```

```
sign-efi-sig-list -g "$(< GUID.txt)" -k PK.key -c PK.crt KEK KEK.esl KEK.auth
```

Se genera el *db.auth* una vez más con la herramienta *openssl* y como common name "my Project":

```
openssl req -newkey rsa:2048 -nodes -keyout db.key -new -x509 -sha256 -days 36500 -subj "/CN=my Signature Database key/" -out db.crt
```

```
openssl x509 -outform DER -in db.crt -out db.cer
```

```
cert-to-efi-sig-list -g "$(< GUID.txt)" db.crt db.esl
```

```
sign-efi-sig-list -g "$(< GUID.txt)" -k KEK.key -c KEK.crt db db.esl db.auth
```

5.4.- Creación del *BOOT* firmado y con herramientas del *TPM*

Para poder construir un *BOOT* con los módulos del *kernel* necesarios, así como de las herramientas necesarias para la descryptación del disco duro, se parte de una imagen base de *ubuntu* dentro de un entorno de *chroot*

```
sudo mount --bind /dev /equipment/dev
sudo mount --bind /run /equipment/run
sudo mount --bind /etc/resolv.conf /equipment/etc/resolv.conf
sudo chroot /equipment
```

Los siguientes pasos se realizan dentro del entorno chroot:

1. Se instala el *kernel* usado por "Enigma" en este caso el 5.3.0-46:

```
apt-get install linux-headers-5.3.0-46 linux-headers-5.3.0-46-
generic linux-image-5.3.0-46-generic linux-modules-5.3.0-46-generic
linux-modules-extra-5.3.0-46-generic
```

Al estar en un entorno *chroot* genera un error la forma de solucionarlo es la siguiente:

```
rm /var/lib/dpkg/info/linux-image-5.3.0-46-generic.postinst -f
dpkg --configure linux-image-5.3.0-46-generic
apt-get install -yf
```

2. Se edita al *fstab* para que el usuario no acceda más que a lo necesario:

```
vim /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to
name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point> <type> <options> <dump> <pass>
/dev/mapper/vg_root-root / squashfs ro,defaults 0 0
# /BOOT was on /dev/mmcblk0p2 during installation
/dev/mapper/vg_root-swap none swap sw 0 0
/dev/mmcblk2p1 /media/card auto
defaults, sync, noauto, x-systemd.automount, x-systemd.device-
timeout=10 0 0
```

3. Se añade *cryptsetup* y sus dependencias a la imagen de *initramfs*

```
sudo vim /etc/cryptsetup-initramfs/conf-hook
CRYPTSETUP=y
```

4. Se edita el fichero `crypttab` para indicar el nombre del volumen, el `device` encriptado y el `script` con el que se descifra

```
sudo vim /etc/crypttab
mmcblk0p3_crypt /dev/mmcblk0p3 none luks,
keyscript=/usr/local/bin/passphrase-from-tpm,initramfs
```

5. Configuramos el sistema como `overlay`:

```
sudo vim /etc/overlayroot.conf
overlayroot_cfgdisk="disabled"
overlayroot="device:dev=/dev/mapper/vg_root-
overlay,swap=1,recurse=0"
```

6. Creamos un `script` para obtener la password del `TPM` con permisos de ejecución:

```
sudo vim /usr/local/bin/passphrase-from-tpm
#!/bin/sh
set -e
export TPM2TOOLS_TCTI="device:/dev/tpm0"
/usr/bin/tpm2_unseal -c 0x81000010 -p pcr:sha1:0,2,3,7
if [ $? -eq 0 ]; then
exit
fi
/lib/cryptsetup/askpass "Unlocking the disk fallback
$CRYPTTAB_SOURCE ($CRYPTTAB_NAME)\nEnter passphrase: "
sudo chmod +x /usr/local/bin/passphrase-from-tpm
```

7. Se añaden al `initramfs` mediante un `hook` las herramientas necesarias para interactuar con el `TPM`:

```
sudo vim /etc/initramfs-tools/hooks/tpm2
#!/bin/bash -e
echo "running tpm2 hook"
if [ "$1" = "prereqs" ]; then exit 0; fi
./usr/share/initramfs-tools/hook-functions
copy_exec /usr/bin/tpm2_unseal
copy_exec /usr/lib/libtss2-tcti-device.so
sudo chmod +x /etc/initramfs-tools/hooks/tpm2
```

8. Se añade como modulo `overlay` al `initramfs`:

```
sudo vim /etc/initramfs-tools/modules
overlay
```

9. Se indica al sistema donde localizar la partición `swap`:

```
sudo vim /etc/initramfs-tools/conf.d/resume
RESUME=/dev/mapper/vg_root-swap
```

10. Se edita el `cmdline` para pasar el Kernel usado:

```
vim /proc/cmdline
BOOT_IMAGE=vmlinuz-5.3.0-46-generic root=/dev/mapper/vg_root-root
ro
update-initramfs -k 5.3.0-46-generic -ut
```

11. Se desmonta el chroot:

```
exit  
sudo umount -lf /equipment/run  
sudo umount -lf /equipment/dev
```

12. Se genera el *BOOTX64.EFI* con los cambios:

```
sudo objcopy --add-section .osrel=/equipment/etc/os-release --  
change-section-vma .osrel=0x20000 --add-section  
.cmdline=/update/proc/cmdline --change-section-vma .cmdline=0x30000  
--add-section .linux="/equipment/BOOT/vmlinuz-5.3.0-46-generic" --  
change-section-vma .linux=0x40000 --add-section .initrd="/equipment/  
BOOT/initrd.img-5.3.0-46-generic" --change-section-vma  
.initrd=0x300000 /usr/lib/systemd/BOOT/efi/linuxx64.efi.stub  
BOOTX64.EFI
```

5.5.- Initrd

Hay varias formas de cargar el fichero de preconfiguración en el instalador de Debian. Durante el desarrollo de la practica se ha optado por la opción de añadir el fichero *preseed* al *initrd.gz* y para ello se ha construido un *script* que no solo almacena en el *initrd.gz* el *preseed*, sino los *scripts* necesarios durante la instalación.

Se opta por la utilización como imagen base la *ISO* de instalación de red *debian:debian-10.7.0-amd-netinst.iso*

```
wget https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/debian-10.7.0-amd64-netinst.iso
```

Para poder extraer los ficheros de la *ISO*:

```
mkdir netinst 7z x -onetinst debian-10.5.0-amd64-netinst.iso
```

Los pasos seguidos para la edición del fichero *initrd* han sido los siguiente:

1. A partir del *initrd* original de la imagen *ISO* descargada se edita el fichero *initrd.gz* para que sea editable:

```
chmod +w -R netinst/install/
```
2. Se descomprime:

```
mkdir -p tmp/initrd
cd tmp/initrd
zcat ../../netinst/install.amd/initrd.gz | cpio -iv
```
3. Y se añaden los ficheros necesarios:

```
cp ../../preseed.cfg .
cp ../../randompass.sh .
cp ../../efiBOOT.sh .
cp ../../script_ch.sh .
cp ../../pcrs.bin .
mkdir -p lib/firmware
cp -a ../../rtl_nic lib/firmware
```
4. Se vuelve a comprimir el *initrd.gz*

```
find . -print0 | cpio -0 -H newc -ov | gzip -c >
../../netinst/install.amd/initrd.gz
chmod -w -R netinst/install
```
5. Y finalmente se regenera el *md5sum.txt*:

```
chmod +w netinst/md5sum.txt
find netinst -follow -type f ! -name md5sum.txt -print0 | xargs -0 md5sum
> netinst/md5sum.txt
chmod -w netinst/md5sum.txt
```

5.5.- Imagen base

Para construir la imagen base que finalmente se instala en el equipo, se parte de un imagen *squashfs* entregada por "Enigmaedia" basada en ubuntu 18.04.

Los pasos a seguir para adaptarlos a la practica son los siguientes:

```
igorTFG sudo unsquashfs systemimage.sqsh
Parallel unsquashfs: Using 4 processors
73394 inodes (75114 blocks) to write

[=====] 75114/75114 100%

created 66169 files
created 11745 directories
created 7218 symlinks
created 7 devices
created 0 fifos
```

Fig 26: Salida comando unsquashfs

1. Se descomprime la imagen mediante la herramienta *unsquashfs* de linux:

```
sudo unsquashfs systemimage.sqsh
```

2. Se cre el fichero "etc/update-motd.d/99-TFG-version", con permisos de ejecución. Dicho fichero se crea como medida para saber que la imagen arrancada por el dispositivo finalizada la instalacion es la correcta:

```
sudo vim squashfs-root/etc/update-motd.d/99-TFG-version
#!/bin/sh
echo "Running Igor TFG"
exit 0
sudo chmod +x squashfs-root/etc/update-motd.d/99-TFG-version
```

3. Se monta el chroot para añadir el usuario UOC:

```
mount --bind /proc squashfs-root/proc
mount --bind /sys squashfs-root/sys
mount --bind /dev squashfs-root/dev
mount --bind /run squashfs-root/run
mount --bind /etc/resolv.conf squashfs-root/etc/resolv.conf
chroot squashfs-root
useradd -m uoc; \
echo uoc2020 | passwd uoc
umount -lf /sys; \
umount -lf /run; \
umount -lf /dev; \
umount -lf /proc; \
exit;
```

4. Se vuelve a comprimir la imagen en squashfs:

```
sudo mksquashfs squashfs-root/ systemimage.sqsh
```



```
igorTFG sudo mksquashfs squashfs-root/ systemimage.sqsh
Parallel mksquashfs: Using 4 processors
Creating 4.0 filesystem on systemimage.sqsh, block size 131072.
[=====] 67889/67889 100%

Exportable Squashfs 4.0 filesystem, gzip compressed, data block size 131072
  compressed data, compressed metadata, compressed fragments, compressed xattrs
  duplicates are removed
Filesystem size 635365.87 Kbytes (620.47 Mbytes)
  35.11% of uncompressed filesystem size (1809437.50 Kbytes)
Inode table size 788864 bytes (770.38 Kbytes)
  27.73% of uncompressed inode table size (2844755 bytes)
Directory table size 792358 bytes (773.79 Kbytes)
  44.20% of uncompressed directory table size (1792534 bytes)
Number of duplicate files found 10321
Number of inodes 85139
Number of files 66169
Number of fragments 4102
Number of symbolic links 7218
Number of device nodes 7
Number of fifo nodes 0
Number of socket nodes 0
Number of directories 11745
Number of ids (unique uids + gids) 21
Number of uids 7
  root (0)
  daemon (1)
  igor (1000)
  unknown (1002)
  syslog (104)
  man (6)
  systemd-resolve (102)
Number of gids 18
  root (0)
  daemon (1)
  igor (1000)
  dip (30)
  shadow (42)
  ssl-cert (112)
  unknown (1002)
  tty (5)
  rdma (105)
  netdev (109)
  uuidd (111)
  crontab (107)
  utmp (43)
  staff (50)
  man (12)
  adm (4)
  systemd-journal (101)
  input (106)
```

Fig 27: Salida comando mksquashfs

5.6.- Imagen ISO arrancable desde un live-CD firmada

Una live-CD se trata de un sistema operativo capaz de arrancar desde un medio extraíble. Con la ayuda de la herramienta de linux *xorriso*, y los ficheros obtenidos en los pasos anteriores, se construye la *live-CD* y se graba en una memoria USB, los pasos serian los siguientes:

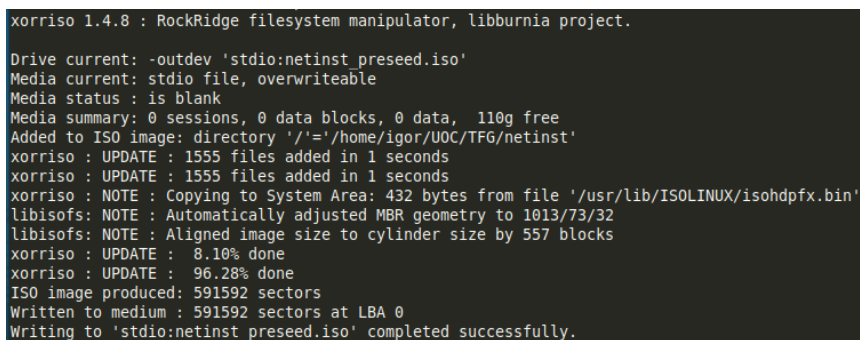
Para firmar el *BOOT* se emplea la herramienta *sbsign* del paquete *sbsigntool* con la cual se firma el *BOOT* de la *ISO* original para que sea reconocido por *EFI* y se procede a su confirmación, el codigo se localiza el fichero `signed_BOOT.sh`:

```
#!/bin/bash

install -d /tmp/BOOTdebian
mount netinst/BOOT/grub/efi.img /tmp/BOOTdebian
sbsign --key certificates/db.key --cert certificates/db.crt --output
/tmp/BOOTdebian/efi/BOOT/BOOTx64.efi /tmp/BOOTdebian/efi/BOOT/BOOTx64.efi
sbsign --key certificates/db.key --cert certificates/db.crt --output
/tmp/BOOTdebian/efi/BOOT/grubx64.efi /tmp/BOOTdebian/efi/BOOT/grubx64.efi
sbsign --key certificates/db.key --cert certificates/db.crt --output
netinst/igor/BOOTX64.EFI netinst/igor/BOOTX64.EFI
sbverify --cert certificates/db.crt /tmp/BOOTdebian/efi/BOOT/grubx64.efi
sbverify --cert certificates/db.crt /tmp/BOOTdebian/efi/BOOT/BOOTx64.efi
sbverify --cert certificates/db.crt netinst/igor/BOOTX64.EFI
```

A fin de generar la *ISO* se ejecuta la herramienta *xorriso* con los parametros adecuados:

```
xorriso -as mkisofs -o netinst_preseed.iso -isohybrid-mbr
/usr/lib/ISOLINUX/isohdpx.bin -c isolinux/BOOT.cat -b
isolinux/isolinux.bin -no-emul-BOOT -BOOT-load-size 4 -BOOT-info-table -
eltorito-alt-BOOT -e BOOT/grub/efi.img -isohybrid-gpt-basdat -no-emul-
BOOT ../netinst
```



```
xorriso 1.4.8 : RockRidge filesystem manipulator, libburnia project.

Drive current: -outdev 'stdio:netinst_preseed.iso'
Media current: stdio file, overwriteable
Media status : is blank
Media summary: 0 sessions, 0 data blocks, 0 data, 110g free
Added to ISO image: directory '/=''/home/igor/UOC/TFG/netinst'
xorriso : UPDATE : 1555 files added in 1 seconds
xorriso : UPDATE : 1555 files added in 1 seconds
xorriso : NOTE : Copying to System Area: 432 bytes from file '/usr/lib/ISOLINUX/isohdpx.bin'
libisofs: NOTE : Automatically adjusted MBR geometry to 1013/73/32
libisofs: NOTE : Aligned image size to cylinder size by 557 blocks
xorriso : UPDATE : 8.10% done
xorriso : UPDATE : 96.28% done
ISO image produced: 591592 sectors
Written to medium : 591592 sectors at LBA 0
Writing to 'stdio:netinst_preseed.iso' completed successfully.
```

Fig 28: Salida comando xorriso

Finalmente, y tras generar la imagen *ISO* se copia en un medio extraíble en este caso *sdb*:

```
sudo cp netinst_preseed.iso /dev/sdb
```

Como medida de seguridad se fuerza la grabación de los datos de la cache:

```
sync
```

En último lugar se desmonta el medio extraíble:

```
sudo umount /media/igor/ISOIMAGE
```

```
7292.145634] usb 3-1.1.1: new high-speed USB device number 6 using xhci_hcd
7292.248425] usb 3-1.1.1: New USB device found, idVendor=1d0d, idProduct=0201
7292.248430] usb 3-1.1.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
7292.248434] usb 3-1.1.1: Product: TDK Trans-it
7292.248437] usb 3-1.1.1: Manufacturer: TDK
7292.248439] usb 3-1.1.1: SerialNumber: 0700078501150483
7292.249640] usb-storage 3-1.1.1:1.0: USB Mass Storage device detected
7292.251611] scsi host9: usb-storage 3-1.1.1:1.0
7293.320854] scsi 9:0:0:0: Direct-Access    TDK          TDK Trans-it    PMAP PQ: 0 ANSI: 0 CCS
7293.322139] sd 9:0:0:0: Attached scsi generic sgl type 0
7294.075520] sd 9:0:0:0: [sdb] 4007936 512-byte logical blocks: (2.05 GB/1.91 GiB)
7294.075802] sd 9:0:0:0: [sdb] Write Protect is off
7294.075807] sd 9:0:0:0: [sdb] Mode Sense: 23 00 00 00
7294.076240] sd 9:0:0:0: [sdb] No Caching mode page found
7294.076250] sd 9:0:0:0: [sdb] Assuming drive cache: write through
7294.142426] sdb: sdb1 sdb2
7294.144551] sd 9:0:0:0: [sdb] Attached SCSI removable disk
7304.819134] ISO 9660 Extensions: RRIP_1991A
TFG git:(master) x sudo cp netinst_preseed.iso /dev/sdb
TFG git:(master) x sync
TFG git:(master) x sudo umount /media/igor/ISOIMAGE
TFG git:(master) x
```

Fig 29: Ejemplo copia de la ISO a USB y posterior umount

5.7.- Instalación

A continuación, se muestran los pasos seguidos durante la instalación y una serie de capturas del proceso.

Para arrancar el sistema de instalación desde el dispositivo USB es necesario presionar F2 y seleccionar el dispositivo.

```
UEFI BOOT:
factory_restore
UEFI: Hard Drive
IP4 Realtek PCIe FE Family Controller
IP6 Realtek PCIe FE Family Controller
UEFI: TDK TDK Trans-it PMAP
OTHER OPTIONS:
BIOS Setup
BIOS Flash Update
Diagnostics
Change Boot Mode Settings
```

Fig 30: BIOS selección UEFI BOOT

Una vez arrancado el instalador de Debian se debe seleccionar la instalación en consola

```
Graphical install
Install
Advanced options ...
Accessible dark contrast installer menu ..
Install with speech synthesis

Enter: Select
E: Edit Selection
C: GRUB Command line
```

Fig 31: Seleccionar durante la instalación consola

Durante la instalación, y como medida de seguridad, se presenta la *password* aleatoria con la que se ha cifrado el disco.

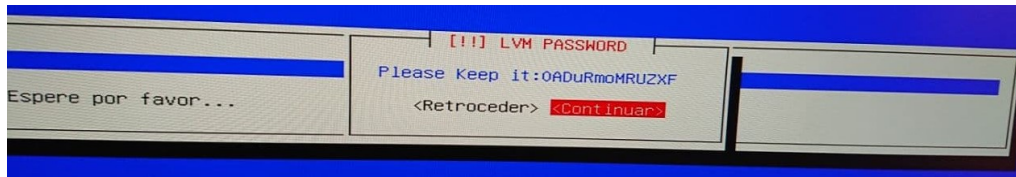


Fig 32: Ejemplo salida password aleatorio

Durante la instalación se puede observar que la *live-CD* instala el "kernel 4.19-10". Dicho *kernel* se sobrescribe posteriormente por el 5.3.0-46 pero es necesario para poder acceder al *TPM* durante la instalación.

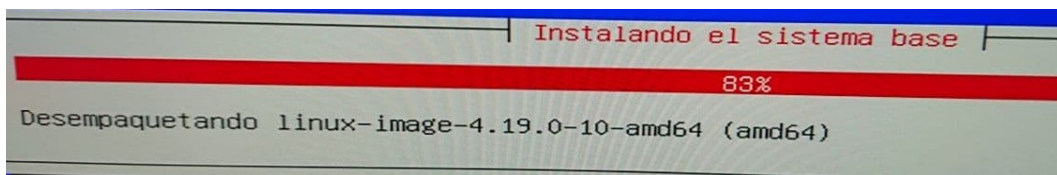


Fig 33: Pantalla que muestra la versión del kernel durante la instalación

Mensaje final de la instalación:

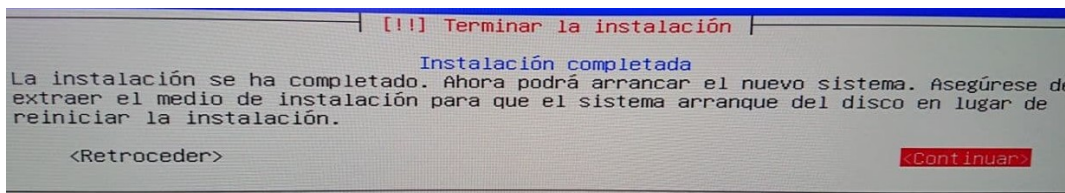
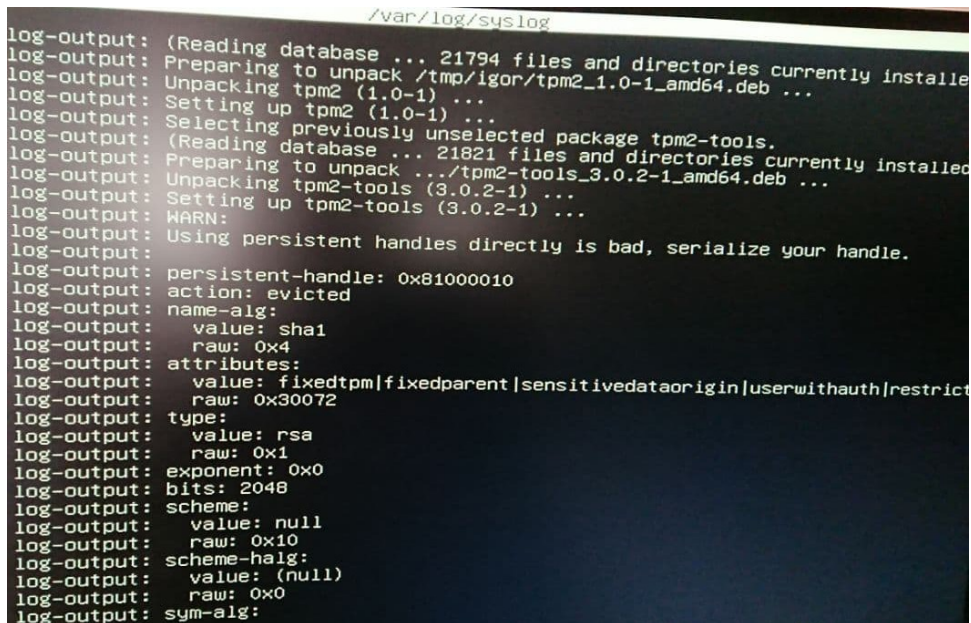


Fig 34: Pantalla final de la instalación

En este momento y con la instalación finalizada se accede al *log* del sistema presionando ALT+F2; lo cual proporciona una consola de *busybox*. Desde dicho *busybox* y con la herramienta que proporciona, *nano*, se accede al *log* (*nano /var/log/syslog*) de la instalación y se presentan algunas capturas.

En la imagen aquí presentada se observa la instalación de los paquetes “.deb” con las herramientas del *TPM* y el acceso al dicho al *TPM*:



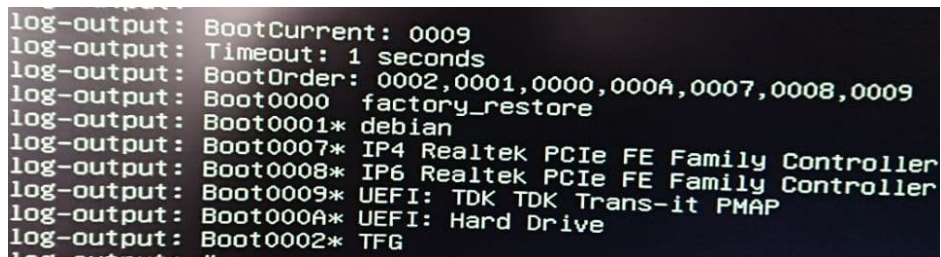
```

/var/log/syslog
log-output: (Reading database ... 21794 files and directories currently installed)
log-output: Preparing to unpack /tmp/igor/tpm2_1.0-1_amd64.deb ...
log-output: Unpacking tpm2 (1.0-1) ...
log-output: Setting up tpm2 (1.0-1) ...
log-output: Selecting previously unselected package tpm2-tools.
log-output: (Reading database ... 21821 files and directories currently installed)
log-output: Preparing to unpack .../tpm2-tools_3.0.2-1_amd64.deb ...
log-output: Unpacking tpm2-tools (3.0.2-1) ...
log-output: Setting up tpm2-tools (3.0.2-1) ...
log-output: WARN:
log-output: Using persistent handles directly is bad, serialize your handle.
log-output:
log-output: persistent-handle: 0x81000010
log-output: action: evicted
log-output: name-alg:
log-output:   value: sha1
log-output:   raw: 0x4
log-output: attributes:
log-output:   value: fixedtpm|fixedparent|sensitivedataorigin|userwithauth|restricted
log-output:   raw: 0x30072
log-output: type:
log-output:   value: rsa
log-output:   raw: 0x1
log-output: exponent: 0x0
log-output: bits: 2048
log-output: scheme:
log-output:   value: null
log-output:   raw: 0x10
log-output: scheme-halg:
log-output:   value: (null)
log-output:   raw: 0x0
log-output: sym-alg:

```

Fig 35: Salida del log durante la instalación con acceso al TPM

En la siguiente imagen refleja como se ha añadido una nueva entrada en el *efiBOOT*:



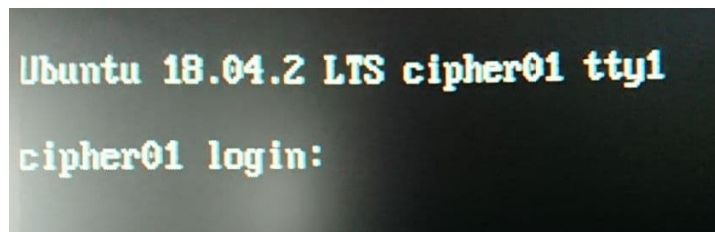
```

log-output: BootCurrent: 0009
log-output: Timeout: 1 seconds
log-output: BootOrder: 0002,0001,0000,000A,0007,0008,0009
log-output: Boot0000 factory_restore
log-output: Boot0001* debian
log-output: Boot0007* IP4 Realtek PCIe FE Family Controller
log-output: Boot0008* IP6 Realtek PCIe FE Family Controller
log-output: Boot0009* UEFI: TDK TDK Trans-it PMAP
log-output: Boot000A* UEFI: Hard Drive
log-output: Boot0002* TFG
log-output: #

```

Fig 36: Salida del log durante la instalación con el cambio de EFI BOOT

Por último, al terminar la instalación y reiniciar el equipo se llega a una consola de *login*.



```

Ubuntu 18.04.2 LTS cipher01 tty1
cipher01 login:

```

Fig 37: Ejemplo consola de login

5.8.- Pruebas realizadas

5.8.1.- Comprobar el sistema.

Una vez instalada la imagen, se comprueba que la imagen arranca correctamente con el *BOOT* adecuado, descifrando el disco duro con la *password* almacenada en el *TPM*. En los recuadros en rojo se puede comprobar que la imagen arranca: con el usuario "UOC", con el *password* "uoc2020" y el "kernel 5.3.0-46".

```

login: uoc
Password:
Welcome to ubuntu 18.04.2 LTS (GNU/Linux 5.3.0-46-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

System information as of Sun Jan 28 17:04:13 CET 2018
System load:  0.01   Memory usage: 8%   Processes:    110
Usage of /home: unknown  Swap usage:  0%   Users logged in: 0

256 packages can be updated.
168 updates are security updates.

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

overlayroot / overlay rw,relatime,lowerdir=/media/root-ro,upperdir=/media/root-rw/overlay,workdir=/media/root-rw/overlay-workdir_0_0
/dev/mapper/ug_root-root /media/root-ro squashfs ro,relatime 0 0
/dev/mapper/ug_root-overlay /media/root-rw ext4 rw,relatime 0 0

Running Igor TFG

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

$ _
  
```

Fig 38: Pantalla de login con el usuario, el kernel 5.3.0-46 y el prompt de "Running Igor TFG"

Además, se ha añadido la entrada "Running Igor TFG" en el fichero "/etc/update-motd.d/99-TFG-version". Se muestra una ampliación de dicho mensaje.

```

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts

overlayroot / overlay rw,relatime,lowerdir=/media/root-ro,upperdir=
/dev/mapper/ug_root-root /media/root-ro squashfs ro,relatime 0 0
/dev/mapper/ug_root-overlay /media/root-rw ext4 rw,relatime 0 0

Running Igor TFG

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
  
```

Fig 39: Pantalla con el prompt "Running Igor TFG"

También, para comprobar que ha ido todo correctamente se comprueba el orden de arranque del *BOOT*

Imagen de la secuencia antes de la instalación:

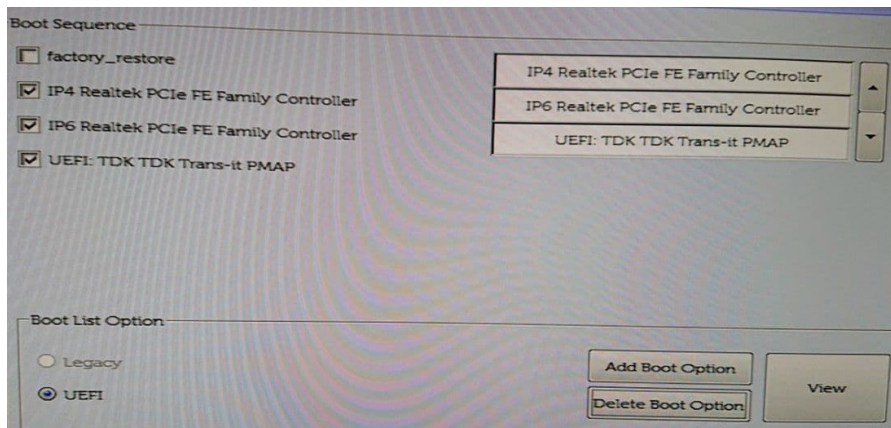


Fig 40: EFI BOOT antes de la instalación

Imagen de la secuencia después de la instalación:

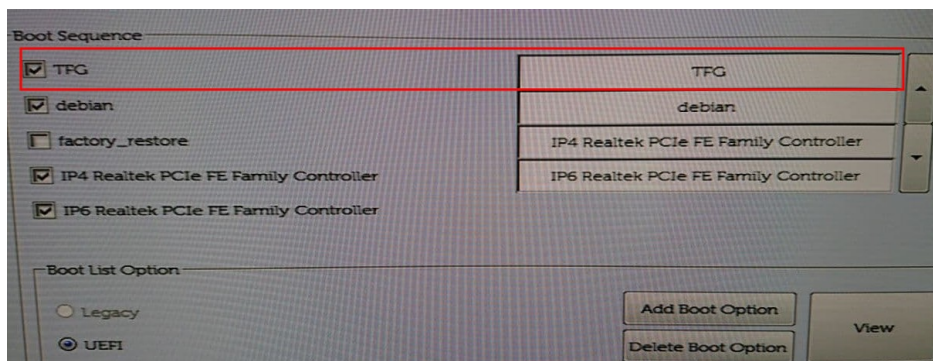


Fig 41: EFI BOOT después de la instalación

5.8.2.- Desactivar *SecureBoot*

Dentro de la batería de pruebas, se muestra que al desactivar el *SecureBoot* los *pcrs* de la maquina no coinciden y al intentar obtener la *password* del *TPM* se produce un error. Las siguientes capturas muestran el proceso de desactivado del *SecureBoot* y el error en los *pcrs*.

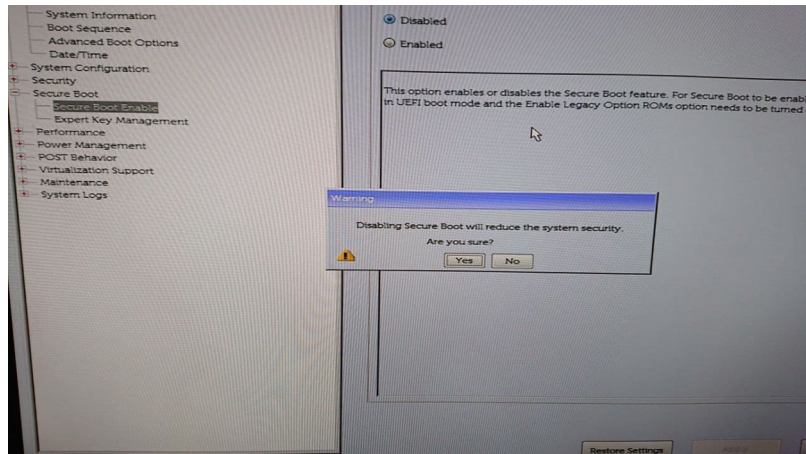


Fig 42: Desactivación del SecureBoot

```

6.4430031 prefetch64-sse: 5728.000 MB/sec
6.4829991 generic_sse: 5078.000 MB/sec
6.4830661 xor: using function: prefetch64-sse (5728.000 MB/sec)
6.4896281 async_tx: api initialized (async)
one.
Begin: Running /scripts/init-precount ... done.
Begin: Mounting root file system ... Begin: Running /scripts/local-top ... WARNING: Failed to connect to
Volume group "vg_root" not found
Cannot process volume group vg_root
WARN: Using persistent handles directly is bad, serialize your handle.
WARNING:esys:src/tss2-esys/api/Esys_Unseal.c:287:Esys_Unseal_Finish() Received TPM Error
ERROR:esys:src/tss2-esys/api/Esys_Unseal.c:94:Esys_Unseal() Esys Finish ErrorCode (0x0000099d)
ERROR: esys_unseal(0xc99d) - tpm:session(1):a policy check failed
ERROR: Unable to run /usr/bin/tpm2_unseal
cryptsetup (nmchl0p3 crypt): cryptsetup failed, bad password or options?
WARN: Using persistent handles directly is bad, serialize your handle.
WARNING:esys:src/tss2-esys/api/Esys_Unseal.c:287:Esys_Unseal_Finish() Received TPM Error
ERROR:esys:src/tss2-esys/api/Esys_Unseal.c:94:Esys_Unseal() Esys Finish ErrorCode (0x0000099d)
ERROR: Esys_Unseal(0xc99d) - tpm:session(1):a policy check failed
ERROR: Unable to run /usr/bin/tpm2_unseal
cryptsetup (nmchl0p3 crypt): cryptsetup failed, bad password or options?
WARN: Using persistent handles directly is bad, serialize your handle.
WARNING:esys:src/tss2-esys/api/Esys_Unseal.c:287:Esys_Unseal_Finish() Received TPM Error
ERROR:esys:src/tss2-esys/api/Esys_Unseal.c:94:Esys_Unseal() Esys Finish ErrorCode (0x0000099d)
ERROR: Esys_Unseal(0xc99d) - tpm:session(1):a policy check failed
ERROR: Unable to run /usr/bin/tpm2_unseal
cryptsetup (nmchl0p3 crypt): cryptsetup failed, bad password or options?
cryptsetup (nmchl0p3 crypt): maximum number of tries exceeded
cryptsetup: going to sleep for 60 seconds...

```

Fig 43: Salida fallo de la política de los pcrs

5.8.3.- USB sin firmar

Otra de las pruebas realizadas es el intento de arranque desde un dispositivo sin firmar. En este caso, el sistema no lo reconoce y no lo arranca. Capturas que muestran dicha prueba:

```
Boot mode is set to: UEFI; Secure Boot: ON

UEFI BOOT:
  TFG
  debian
  factory_restore
  IP4 Realtek PCIe FE Family Controller
  IP6 Realtek PCIe FE Family Controller
  UEFI: KingstonDataTraveler2.0 1.00
OTHER OPTIONS:
  BIOS Setup
  BIOS Flash Update
  Diagnostics
  Change Boot Mode Settings
```

Fig 44: UEFI BOOT con USB sin firmar

```
Operating System Loader signature not found in SecureBoot database ('db').

Press F1 key to retry boot.
Press F2 key to reboot into setup.
Press F5 key to run onboard diagnostics.
```

Fig 45: Salida de firma no localizada en el SecureBoot

6.- Conclusiones y continuidad del trabajo realizado

La principal conclusión obtenida tras finalizar el proyecto es que en definitiva y tras un amplio trabajo de investigación ha sido posible cumplir los objetivos iniciales del proyecto.

Cabe destacar la tareas de investigación que han supuesto un verdadero reto, debido principalmente a la diversidad de tecnologías usadas durante el desarrollo del proyecto y al uso novel de ellas. Pese a disponer de información de todas ellas ha sido todo un reto el poder hacerlas trabajar conjuntamente.

A todo ello, hay que sumar la complejidad que supone la gestión en un proyecto real. La auto planificación y poder estimar correctamente los plazos, cuando hay tareas de investigación e incertidumbre, han supuesto un mejor conocimiento de uno mismo, al mismo tiempo de las capacidades y de las tareas que se pueden afrontar.

Aun así y como se ha visto durante el desarrollo de la practica, se han cometido errores de confianza que han supuesto pequeñas desviaciones en las fechas estimadas. El poder aclarar las incertidumbres, para poder obtener los objetivos, han supuesto más horas que las planificadas. A pesar de ello ha sido posible finalizar el proyecto en plazo.

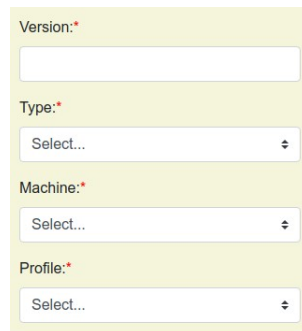
Con un poco de objetividad y una vez finalizado el proyecto, se podría decir que a nivel personal ha servido para poder aplicar los conocimientos adquiridos, así como, la oportunidad de investigar nuevas tecnologías: como gestión de certificados, imágenes *squashfs*, etc. Del mismo modo ha permitido entender mejor el proceso de arranque de Linux, junto a la creación de una *Live-CD* y los pasos seguidos durante una instalación.

Finalmente, y como objetivo adicional se ha conseguido la integración universidad-empresa, permitiendo obtener las bases para un futuro proyecto dentro de la empresa "Enigmedia", cuyo desarrollo se pondrá en marcha en el futuro próximo.

6.1.- Próximos pasos

Con el objetivo de tener un proyecto acabado dentro de la empresa “Enigmedia”, los líneas de futuro a implementar serían:

- 1.- Debido a que el código se trata de una prueba de concepto. Lo primero que se debería realizar es la mejora del código y su optimización.
- 2.- Como mejora del proyecto se podría establecer un pequeño formulario para poder seleccionar la imagen base. Esto facilita la posibilidad que con la misma imagen *ISO* se pueda realizar la instalación en distintos tipos de *hardware* y con distintos perfiles como desarrollo y producción.



Version:*

Type:*

Machine:*

Profile:*

Fig 46: Mockup formulario de instalación

- 3.- En función del formulario seleccionar la imagen base y descargarla de un repositorio, durante la instalación. Esto brinda la posibilidad de aligerar la *ISO* y las actualizaciones de las imágenes base.

7.- Glosario

BootLoader: Gestor de arranque que se encarga de realizar diversas pruebas antes del arranque del sistema operativo, así como de dar instrucciones a este para que el arranque se realice sin problemas.

Busibox: Se trata de un software suite que proporciona varias utilidades Unix en un solo ejecutable.

CA: En certificados, se trata de una entidad de confianza, encargado de emitir y revocar los certificados, utilizando en ellos la firma electrónica, para lo cual se emplea la clave pública.

Certificado: Fichero firmado eléctricamente, por un prestador de servicios de certificación, considerado por otras entidades como una autoridad, que asocia unos datos de verificación de firma a un firmante, de forma que exclusivamente pueda firmar este firmante y confirma su identidad.

Checksum: O suma de verificación, es una función de redundancia cuyo objetivo principal es detectar cambios accidentales en una secuencia de datos para proteger la integridad de estos.

Chroot: Herramienta de linux por la que se cambia el directorio *root* del disco (y el actual proceso y sus hijos) a otro directorio *root* simulado.

Cifrado: Dentro de la criptografía, el cifrado se trata de un procedimiento que utiliza un algoritmo de cifrado con cierta clave para transformar un mensaje, de tal forma que sea incomprensible o, al menos, difícil de comprender sin disponer de la clave secreta.

Commom Criteria: O criterios comunes para la valoración de la seguridad de la tecnología de la información. Ámbito en el que los usuarios de sistemas informáticos pueden establecer sus requisitos funcionales y de garantía de seguridad.

Cpio: Se basa en una serie de ficheros y directorios tanto los encabezados, utilizados por GNU CPIO para extraer el archivo, al mismo tiempo encabezados extra como el nombre, fecha de creación, permisos y propietario de cada fichero y directorio.

CRL: Se trata de un registro para mantener una lista de certificados que ya no son válidos y en los que no se debería confiar.

Debian: O Proyecto *Debian*, es una comunidad conformada por desarrolladores y usuarios, que mantiene un sistema operativo *GNU* basado en software libre.

Drivers: Programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del *hardware* y proporcionar una interfaz para utilizar el dispositivo.

eMMC: Son un tipo de memorias *NAND flash* de corta capacidad y reducido tamaño y precio.

Gzip: Abreviatura de GNU ZIP. Es un algoritmo libre y de código abierto para la compresión de archivos.

Hash: En criptografía se trata de un algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija. Independientemente de la longitud de los datos de entrada, el valor *hash* de salida tendrá siempre la misma longitud.

Hook: Técnica utilizada para alterar o aumentar el comportamiento de un sistema operativo, de aplicaciones o de otros componentes de software interceptando llamadas de función o mensajes entre componentes de software.

Initrd: Es un sistema de ficheros temporal usado por el núcleo de Linux durante el inicio del sistema.

Initramfs: Se trata de un sistema de archivos *RAM* de inicio, fundamentado en *tmpfs*, con el objetivo de montar el sistema de archivos raíz mínimo para permitir la ejecución de programas antes de que se monte el verdadero sistema de archivos raíz.

ISO: Una imagen *ISO* es un archivo informático donde se almacena una copia o imagen exacta de un sistema de archivos. Se rige por el estándar *ISO 9660*, que le da nombre.

LINCE: Metodología de Evaluación para la Certificación Nacional Esencial de Seguridad.

LiveCD: Se trata de un sistema operativo capaz de arrancar desde un medio extraíble.

LVM: Es una implementación de un gestor de volúmenes lógicos para el núcleo de Linux.

LUKS: De las siglas en inglés, Linux Unified Key Setup, es una especificación de cifrado de disco creada por Clemens Fruhwirth.

MBR: Registro de arranque principal, también conocido como registro de arranque maestro (en inglés *master boot record*), es el primer sector del disco duro.

Nonce: En criptografía, un nonce es un número arbitrario que puede ser utilizado una sola vez en una comunicación criptográfica. Suele ser un número aleatorio o pseudoaleatorio.

OverlayFS: Implementación de sistema de archivos de montaje de unión para Linux. Combina varios puntos de montaje subyacentes diferentes en uno solo, lo que da como resultado una estructura de directorio única que contiene archivos y subdirectorios subyacentes de todas las fuentes.

Partman: Herramienta recomendada en Debian para particionar discos. También puede ajustar el tamaño de las particiones, crear sistema de ficheros y asignarlos a sus respectivos puntos de montaje.

PCR: Los registros de *PCR* o de configuración de la plataforma son objetos especiales de *TPM* que sólo pueden modificarse o escribirse mediante un mecanismo de extensión de *hash*.

Preseed: Fichero de configuración que permite realizar instalaciones de forma automática o desatendida.

PoE: Tecnología que incorpora alimentación eléctrica a una infraestructura LAN estándar. Permite que la alimentación eléctrica se suministre a un dispositivo de red usando el mismo cable que se utiliza para la conexión de red.

Rock Ridge: Es el nombre de unos conjuntos de información adicional que mejora un sistema de archivos *ISO 9660* para que pueda representar un sistema de archivos compatible: con *POSIX*, permisos de acceso, enlaces simbólicos y otros atributos.

SecureBoot: *UEFI Secure boot* es un mecanismo de verificación para asegurar que el código lanzado por el *firmware* es confiable.

Script: Secuencia de comandos que se usa para designar a un programa relativamente simple.

Squashfs: Sistema de archivos comprimidos de solo lectura para Linux.

TPM: Módulo de plataforma de confianza, se trata de una especificación publicada que detalla un criptoprocesador seguro que puede almacenar claves de cifrado para preservar información.

Ubuntu: Es un sistema operativo de software libre y código abierto basada en Debian.

UEFI: La *Unified Extensible Firmware Interface* o interfaz de *firmware* extensible unificada, es una especificación que detalla una interfaz entre el sistema operativo y el *firmware*.

Volúmenes RAID: *Redundant Array of Independent Disks* o matriz de discos independientes redundantes, se trata de una técnica de redundancia por la cual se usa dos o más discos para almacenar los datos.

x.509: Es un estándar para infraestructuras de claves públicas que especifica formatos estándar para certificados de claves públicas y un algoritmo de validación de la ruta de certificación.

8.- Índice de figuras

Índice de Figuras

Fig 1: Dell descripción física.....	7
Fig 2: Dell Dimensiones y Peso.....	8
Fig 3: Cubic.....	16
Fig 4: Systemback.....	17
Fig 5: SecureBoot.....	20
Fig 6: LVM de la práctica.....	21
Fig 7: TPM.....	23
Fig 8: Proceso de arranque de linux.....	26
Fig 9: Squashfs.....	28
Fig 10: OverlayFS.....	30
Fig 11: Versión chroot.....	31
Fig 12: Versión mksquashfs y unsquashfs.....	31
Fig 13: Versión OpenSSL.....	32
Fig 14: Versión sbsign y sbverify.....	32
Fig 15: Versión xorriso.....	32
Fig 16: Versión efiBOOTmgr.....	33
Fig 17: Versión dd.....	33
Fig 18: Versión checlinstall.....	33
Fig 19: Versión objcopy.....	34
Fig 20: BIOS selección Expert Key Management.....	35
Fig 21: BIOS selección Enable Custom Mode.....	35
Fig 22: BIOS selección PK.auth.....	36
Fig 23: BIOS selección db.auth.....	36
Fig 24: BIOS selección KEK.auth.....	36
Fig 25: Particionado durante la instalación.....	37
Fig 26: Salida comando unsquashfs.....	48
Fig 27: Salida comando mksquashfs.....	49
Fig 28: Salida comando xorriso.....	50
Fig 29: Ejemplo copia de la ISO a USB y posterior umount.....	51
Fig 30: BIOS seleccion UEFI BOOT.....	52
Fig 31: Seleccionar durante la instalación consola.....	52
Fig 32: Ejemplo salida password aleatorio.....	53
Fig 33: Pantalla que muestra la versión del kernel durante la instalación.....	53
Fig 34: Pantalla final de la instalación.....	53
Fig 35: Salida del log durante la instalación con acceso al TPM.....	54
Fig 36: Salida del log durante la instalación con el cambio de EFI BOOT.....	54
Fig 37: Ejemplo consola de login.....	54
Fig 38: Pantalla de login con el usuario, el kernel 5.3.0-46 y el prompt de "Running Igor TFG".....	55
Fig 39: Pantalla con el prompt "Running Igor TFG".....	55
Fig 40: EFI BOOT antes de la instalación.....	56

Fig 41: EFI BOOT después de la instalación.....	56
Fig 42: Desactivación del SecureBoot.....	57
Fig 43: Salida fallo de la politica de los pcrs.....	57
Fig 44: UEFI BOOT con USB sin firmar.....	58
Fig 45: Salida de firma no localizada en el SecureBoot.....	58
Fig 46: Mockup formulario de instalación.....	60

9.- Bibliografía

- [https://www.linux-magazine.com/Issues/2014/164/The-State-of-Secure-BOOT/\(offset\)/3](https://www.linux-magazine.com/Issues/2014/164/The-State-of-Secure-BOOT/(offset)/3)
- <https://gitlab.com/cryptsetup/cryptsetup/blob/master/README.md>
- <https://wiki.gentoo.org/wiki/Initramfs/Guide/es>
- <https://debian-handbook.info/browse/da-DK/stable/unix-services.html>
- <https://link.springer.com/book/10.1007/978-1-4302-6584-9>
- <https://www.kernel.org/doc/html/latest/filesystems/squashfs.html>
- <https://www.kernel.org/doc/html/latest/filesystems/overlayfs.html>
- <https://wiki.debian.org/es/LVM>
- https://www.uv.es/sto/articulos/BEI-2003-11/certificados_digitales.html
- [https://wiki.archlinux.org/index.php/Chroot_\(Español\)](https://wiki.archlinux.org/index.php/Chroot_(Español))
- http://cv.uoc.edu/tren/trenacc/web/GAT_EXP.PLANDOCENTE?any_academico=20201&cod_asignatura=75.601&idioma=CAS&pagina=PD_PRE_V_PORTAL
- <https://www.debian.org/releases/stretch/mips/index.html.es>
- <https://www.debian.org/releases/stable/i386/apb.en.html>
- <https://threat.tevora.com/secure-BOOT-tpm-2/>
- <https://github.com/tpm2-software/tpm2-tools>
- https://wiki.archlinux.org/index.php/Unified_Extensible_Firmware_Interface/Secure_BOOT
- <https://es.wikipedia.org/>