



Curso Multimedia Home Platform 1.1.2

Security. Unsigned vs Signed Apps

Signed & Unsigned APPS

Fichero Externo de Permisos

Firma de Aplicaciones

Curso Multimedia Home Platform 1.1.2

Copyright 2008 © Enrique Pérez Gil

Licensed under the ***Creative Commons Attribution-Non-Commercial-No Derivative Works 3.0 Unported License***. You may not use this file except in compliance with the License. You may obtain a copy of the License at:

<http://creativecommons.org/licenses/by-nc-nd/3.0/legalcode>

This is a human-readable summary of the License applied:

(<http://creativecommons.org/licenses/by-nc-nd/3.0/>)

You are free to Share, to copy, distribute and transmit the work **Under the following conditions:**

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **Noncommercial.** You may not use this work for commercial purposes.
- **No Derivative Works.** You may not alter, transform, or build upon this work.

For any reuse or distribution, you must make clear to others the license terms of this work. Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.

Introducción

- El contexto en el que estamos es un tanto “paranoico”, tened en cuenta que si falla el software del deco ¿ cuantos le van a echar la culpa a tu aplicación y cuantos a Philips ?
- Estamos obligados a hacer aplicaciones sólidas, la TV lleva funcionando sólidamente muchos años, de forma que si no lo hacemos así, se perdería la confianza en la TV Interactiva.
- Desde el punto de vista de la seguridad tenemos tres escenarios fundamentales a considerar:
 - Queremos que el personal vea sólo aquello que está autorizado a ver.
 - Queremos que el contenido que llegue sea el que se emite. Esto en cuanto a las aplicaciones tiene especial relevancia: no queremos que nadie pueda cambiar una clase por otra para hacer cualquier “barbaridad” como por ejemplo marcar un tf 906...
 - Además queremos que las aplicaciones puedan hacer sólo aquello que se les permite. Por ejemplo sólo si se tiene permiso se puede acceder al canal de retorno.

Introducción

- Como veremos en detalle, el modelo de seguridad establecido permite **a los operadores** establecer qué puede hacer o no una aplicación, como por ejemplo a qué servidores acceder por el Return Channel; mediante este tipo de configuración ahorramos mucho testing y por lo tanto añadimos fiabilidad.
- Fundamentalmente el nivel de acceso de una aplicación se basa en:
 - los permisos que da el Broadcaster y
 - las preferencias del usuario.

El Modelo de Permisos. Permissions

- Gran parte del modelo de seguridad se basa en el de J2SE, en el sentido de que por defecto las apps no podrán acceder a numerosos APIS lanzándose las `SecurityException` oportunas cuando se intente; por ejemplo, las apps por defecto no pueden leer ciertas propiedades del Sistema, o usar el Return Channel o el Persistent Storage...
- Para describir lo que se puede hacer el sistema se apoya mayormente en los **Permission Objects** de cada API; de esta forma es posible definir con mucho detalle a qué funcionalidades se tiene acceso y a cuales no.

El Modelo de Permisos. Permissions

- MHP 1.1.2 Specs: “**sandbox**: unsigned applications and signed applications **without a permission file** have access to all the APIs for which **there is no permission signalling defined**, this is commonly called the sandbox”
- Dicho de otra forma: existen unas restricciones y servicios accesibles por defecto que se aplican/ofrecen a las apps sin firmar y también a las firmadas que no lleven el **Permission Request File**.
- Importante: Los objetos **Permission** se configuran por el entorno para gestionar a nivel de API Java el nivel de acceso a los servicios, **lo cual a nosotros no nos afecta desde el punto de vista de desarrollo: es una “herramienta” del entorno**. Es en el Permission Request File donde sí que nosotros (o el broadcaster) indicamos qué se desea hacer, y es mediante la configuración por parte del sistema de objetos Permission como se “da permiso” para hacerlo o no.

- Antes de proseguir, una signed application es aquella que está firmada siguiendo los procesos clásicos de claves públicas/privadas, certificados...etc. Veremos el detalle de cómo se firman más adelante.
- Además, tal y como dicen las MHP 1.1.2 specs A0068r1 en 12.6.1, TODAS las clases de que consta la Aplicación habrán de estar firmadas, (con independencia de su procedencia):

For a DVB-J application to be correctly authenticated, **all the class files that the application consists of** need to be signed, the signatures need to verify (see clause 12.4.4) and the application_id needs to be from within the range allocated to signed applications (see Table 77). If, during the loading or execution of the application the MHP detects a signed file containing a class that failed to pass the authentication process (e.g. because its actual hash value does not match the expected hash value), then the class shall be considered as not available.

Permisos para aplicaciones NO Firmadas

- Veamos la relación entre los apis y los Permission. Los que se van a describir son los que hay.

java.awt.AWTPermission	Permiso para acceder a partes sensibles de la AWT. Denegado para signed y unsigned apps.
java.net.SocketPermission	No está en el sandbox la operativa de Sockets, luego SocketPermission no es necesario configurarlo. Curioso: con Security se puede alimentar un HSound con una http URL
java.util.PropertyPermission	Se da permiso de lectura para leer todas las propiedades definidas en el doc salvo para aquellas que son accesibles solo para signed apps. No se da permiso de “write” en ningún caso. Ved Exercise_ciclo6 error security.txt o Exercise_sec1.txt donde podréis ver qué properties se pueden ver y cuales no.
java.lang.RuntimePermission	Denegado para signed y unsigned
java.io.SerializablePermission	Denegado para signed y unsigned. Se refiere a la extensión de ObjectOutputStream u ObjectInputStream, o bien a la posibilidad de sustituir un objeto por otro cuando se serializa.

Permisos para aplicaciones NO Firmadas (y 2)

java.io.FilePermission	De lectura para el directorio donde se monta la aplicación y los object carousels
javax.tv.media.MediaSelectPermission	Está en el sandbox. Se da acceso para poder seleccionar todos los streams: "*". Recordemos el MediaSelectControl que nos permitía elegir los Streams que deseábamos ver en un Service.
javax.tv.service.ReadPermission	Está en el sandbox. Se da acceso para leer Locators de tipo: *
javax.tv.service.selection.ServiceContextPermission	Se da con el nombre getServiceContentHandlers y "own"
java.util.Locale.setDefault	Lanzará una Exception
Aplicaciones Signalled a través del Return Channel (10.4.9)	<p>Dispondrán de un java.security.SocketPermission permitiendo connect para todos los pares de host:port definidos. Es decir, tienen acceso a aquellos host:port de los que se bajaron. Los Interactive Channels se ven en un capítulo dedicado.</p> <p>Applications signalled in an AIT file clause 10.4.9 shall have an instance of SocketPermission for each combination of host and port number listed in all transport protocol descriptors with the protocol ID value 0x0003 in both their AIT entry and in the "common" descriptor loop. Each permission shall have the "connect" action only.</p>

Permisos adicionales para aplicaciones Firmadas

- Recordemos que no se tiene acceso a estas operaciones por defecto, salvo allí donde se indique, sino que se pueden solicitar mediante el **Permission Request File. Igualmente insistir en que la aplicación debe estar FIRMADA. A continuación vemos la relación con los Permission**

java.util.PropertyPermission	Se da acceso por defecto a dvb.persistent.root
java.io.FilePermission	Se crea cuando el PRF solicita permiso para acceder al “persistent storage” o bien para acceder a memory cards y este se otorga. También se creará cuando se encuentran credenciales para acceder a un determinado fichero.
org.dvb.net.ca.CAPermission	Se crea cuando el PRF (Permission Request File) solicita comunicación con el CA subsystem y este se otorga
org.dvb.application.AppsControlPermission	Se crea cuando el PRF solicita permiso para manejar con mayores posibilidades el ciclo de vida de las apps y este se otorga.
org.dvb.net.rc.RCPermission	Se crea cuando el PRF solicita permiso para comunicarse a través del Return Channel y este se otorga. (para conexiones TF sobre todo).
org.dvb.net.tuning.TunerPermission	Se crea cuando el PRF solicita permiso para acceder al Tuning API y este se otorga

Permisos adicionales para aplicaciones Firmadas (y 2)

javax.tv.service.selection.SelectPermission	Se crea por defecto con locator * y "own". También se crea un ServiceContextPermission de igual forma.
org.dvb.user.UserPreferencePermission	Se crea cuando el PRF solicita permiso para leer/modificar las preferencias de usuario y este se otorga.
java.net.SocketPermission	Se crea cuando el PRF solicita permiso para comunicarse con un host:port y este se otorga. Este permiso no se otorgará cuando todos los Return Channels esté representados por org.dvb.net.rc.ConnectionRCInterface y no exista al menos un org.dvb.net.rc.RCPermission
org.dvb.media.DripFeedPermission	Se crea cuando el PRF solicita permiso y este es otorgado para trabajar con un dripfeed
org.dvb.application.storage.ApplicationStoragePermission	Se crea cuando el PRF solicita permiso y este es otorgado para almacenar apps
javax.microedition.apdu.APDUPermission	Se crea cuando el PRF solicita permiso y este es otorgado para comunicarse con el non CA smartcard API

Permisos adicionales para aplicaciones Firmadas (y 3)

<code>javax.tv.service.Selection.ServiceContextPermission</code>	Por defecto, en Internet Profile, todas aquellas apps que tengan <code>javax.tv.service.selection.SelectPermission</code> tendrán permisos para crear, destruir y parar <code>ServiceContexts</code> lo cual se configura mediante la instanciación de <code>ServiceContextPermission</code> con dichas opciones para "own".
<code>javax.microedition.xlet.ixc.IxcPermission</code>	Se ofrecerá por defecto la forma (idem a no firmadas): <ul style="list-style-type: none">•<code>IxcPermission("dvb:/ixc/organisation_id/application_id/*", "bind")</code>•<code>IxcPermission("dvb:/ixc/*", "lookup")</code>

Más adelante veremos en detalle las políticas de seguridad con respecto a los diferentes aspectos de los APIS.

Permission Request File: ¿ cómo es ? DTD MHP 1.1

```
<!ELEMENT permissionrequestfile (file?,capermission?,applifecyclecontrol?,returnchannel?,tuning?,servicesel?,userpreferences?,network?,dripfeed?, persistentfilecredential*, applicationstorage?, smartcardaccess?,providerpermission?)>
<!ATTLIST permissionrequestfile orgid CDATA #REQUIRED appid CDATA #REQUIRED>
<!ELEMENT file EMPTY>
<!ATTLIST file value (true|false) "true">
<!ELEMENT capermission (casystemid)+>
<!ELEMENT casystemid EMPTY>
<!ATTLIST casystemid entitlementquery (true|false) "false" id CDATA #REQUIRED mmi (true|false) "false" messagepassing (true|false) "false" buy (true|false) "false">
<!ELEMENT applifecyclecontrol EMPTY>
<!ATTLIST applifecyclecontrol value (true|false) "true">
<!ELEMENT returnchannel (defaultisp?,phonenumber*)>
<!ELEMENT defaultisp EMPTY>
<!ELEMENT phonenumber (#PCDATA)>
<!ELEMENT tuning EMPTY>
<!ATTLIST tuning value (true|false) "true">
<!ELEMENT servicesel EMPTY>
<!ATTLIST servicesel value (true|false) "true">
<!ELEMENT userpreferences EMPTY>
<!ATTLIST userpreferences write (true|false) "false" read (true|false) "true">
<!ELEMENT network (host)+>
<!ELEMENT host (#PCDATA)>
<!ATTLIST host action CDATA #REQUIRED>
```

Permission Request File: ¿ cómo es ? DTD MHP 1.1 (y 2)

```
<!ELEMENT dripfeed EMPTY>
<!ATTLIST dripfeed value (true|false) "true">
<!ELEMENT persistentfilecredential (grantoridentifier,expirationdate,filename+,signature,certchainfileid)>
<!ELEMENT grantoridentifier EMPTY>
<!ATTLIST grantoridentifier id CDATA #REQUIRED>
<!ELEMENT expirationdate EMPTY>
<!ATTLIST expirationdate date CDATA #REQUIRED>
<!ELEMENT filename (#PCDATA)>
<!ATTLIST filename write (true|false) "true" read (true|false) "true">
<!ELEMENT signature (#PCDATA)>
<!ELEMENT certchainfileid (#PCDATA)>
<!-- new elements in MHP 1.1 -->
<!ELEMENT applicationstorage (applicationstorageorg)*>
<!ATTLIST applicationstorage manageservice (true|false) "false" createservice (true|false) "false" deleteservice (true|false) "false" managecache (true|false) "false">
<!ELEMENT applicationstorageorg EMPTY>
<!ATTLIST applicationstorageorg orgid CDATA #REQUIRED storeservice (true|false) "false" removeservice (true|false) "false" storecache (true|false) "false" removecache (true|false) "false">
<!ELEMENT smartcardaccess EMPTY>
<!ELEMENT privilegedrce EMPTY>
<!ATTLIST privilegedrce value (internal|external) "internal">
<!ELEMENT provider (name)+ >
<!ELEMENT name (#PCDATA)>
<!ATTLIST name scope (application|system) "application">
```

Permission Request File: ¿ cómo es ? Ejemplo

```
<?xml version="1.0"?>
<!DOCTYPE permissionrequestfile PUBLIC "-//DVB//DTD
Permission Request File 1.1//EN"
"http://www.dvb.org/mhp/dtd/permissionrequestfile-1-1.dtd">
<permissionrequestfile orgid="0x23d2" appid="0x4020">
<file value="true"></file>
<capermission>
<casystemid id="0x1111" messagepassing="true"
entitlementquery="true" mmi="false">
</casystemid>
</capermission>
<applifecyclecontrol value="true"></applifecyclecontrol>
<returnchannel>
<defaultisp></defaultisp>
<phonenumber>+3583111111</phonenumber>
<phonenumber>+3583111112</phonenumber>
<phonenumber></phonenumber>
</returnchannel>
<tuning value="false"></tuning>
<servicesel value="true"></servicesel>
```

```
<userpreferences read="true" write="false"> </userpreferences>
<network>
<host action="connect">com.kke.cz:8893</host>
</network>
<persistentfilecredential>
<grantoridentifier id="0x05"></grantoridentifier>
<expirationdate date="24/12/2032"></expirationdate>
<filename read="true" write="false">5/15/dir1/scores</filename>
<filename read="true" write="false">5/15/dir1/names</filename>
<signature>023203293292932921493143929423943294239432
</signature>
<certchainfileid>3</certchainfileid>
</persistentfilecredential>
<provider>
<name scope="application">0x00000B.
EMV_PK11.VISA_REVOLVER</name>
</provider>
<smartcardaccess/>
</permissionrequestfile>
```

- El PRF se almacena en el mismo directorio en el que reside el XLET y tiene el nombre: `dvb.xlet_name.perm`

Permission Request File: orgid appid

```
<!ATTLIST permissionrequestfile orgid CDATA #REQUIRED appid CDATA #REQUIRED>
```

- Consisten en el appid y el orgid en Hex.

```
<permissionrequestfile orgid="0x23d2" appid="0x4020">
```

Permission Request File: File access al PersistentStorage

```
<!ELEMENT file EMPTY>
```

```
<!ATTLIST file value (true|false) "true">
```

- Se solicita acceso al **persistent Storage**. **Sólo para signed**. El Directorio raíz es el especificado por la property: **dvb.persistent.root** y los directorios a los que tendrá acceso serán los dos siguientes (no se deja claro a cual):
 - PROP(dvb.persistent.root)/orgid
 - PROP(dvb.persistent.root)/orgid/appid

```
<file value="true"></file>
```

El API de Persistent Storage en detalla en un capítulo específico (dvb.persistent.root...)

Permission Request File: CA API

- Las unsigned NO tienen acceso a los siguientes métodos:

org.davic.net.ca.CAModule.buyEntitlements.

org.davic.net.ca.CAModule.openMessageSession.

org.davic.net.ca.CAModuleManager.addMMIListener.

org.davic.net.ca.CAModule.queryEntitlements.

org.davic.net.ca.CAModule.listEntitlements.

- Las signed pueden solicitarlo en el PRF de la forma:

```
<!ELEMENT capermission (casystemid)+>
```

```
<!ELEMENT casystemid EMPTY>
```

```
<!ATTLIST casystemid entitlementquery true|false "false" id CDATA #REQUIRED mmi (true|false) "false" messagepassing (true|false) "false"
```

```
buy (true|false) "false" >
```

Los CA ID tienen el formato:

```
CAIds = CASystemId | "[" CASystemId "-" CASystemId "]" | ""
```

```
CASystemId = "0x" 4hex
```

Ejemplo:

```
<capermission>
```

```
<casystemid id="0x1111" messagepassing="true" entitlementquery="true" mmi="false"> </casystemid>
```

```
</capermission>
```

Permission Request File: Apps Life Cycle Control

- Las unsigned sólo pueden lanzar las visibles en el APP Listing API provisto por el Service actual para el ServiceContext donde se encuentra la app que desea ejecutarla.
- Una unsigned SOLO puede controlar el ciclo de vida de una app que ha lanzado ella: start, pause, stop, resume
- Una signed dispone de los mismos derechos que una unsigned salvo que se solicite en el PRF que se quiere poder controlar el ciclo de vida de TODAS las apps signalled, tanto si las ha lanzado ella como si no. (Podría ser un Launcher)

<!ELEMENT applifecyclecontrol EMPTY>

<!ATTLIST applifecyclecontrol value (true|false) "true" >

Ejemplo: <applifecyclecontrol value="true"></applifecyclecontrol>

El App Listing & Launching API se describe en un capítulo específico

Permission Request File: Return Channel (VIA ISP, phone-number)

- Las unsigned **no** tienen acceso mientras que las signed pueden solicitarlo de la siguiente forma, en la cual se especifican los números de TF a marcar.

```
<!ELEMENT returnchannel (defaultisp?,phonenum*)>
```

```
<!ELEMENT phonenum (#PCDATA)>
```

```
<!ELEMENT defaultisp EMPTY> (sirve para indicar que se puede usar el default).
```

Ejemplo:

```
<returnchannel>
```

```
<defaultisp></defaultisp>
```

```
<phonenum>+3583111111</phonenum>
```

```
<phonenum>+3583111112</phonenum>
```

```
<phonenum></phonenum>
```

```
</returnchannel>
```

- Interesante: si se deja el phonenum vacío se podría llamar a cualquier número, aunque como veremos en el API, **el usuario será preguntado antes**.

Este API se ve en detalle en un capítulo específico: org.davic.net.rc

Permission Request File: Tuning API

- Las Unsigned NO pueden usar el Tuning API. Las signed pueden solicitarlo en el PRF de la forma siguiente:

```
<!ELEMENT tuning EMPTY><
```

```
<!ATTLIST tuning value (true|false) "true" >
```

Ejemplo: `<tuning value="false"></tuning>`

El Tuning API se ve en detalle en un capítulo específico: [org.davic.net.tuning](http://org.davic.net/tuning)

Permission Request File: Service Selection

- Las Unsigned NO pueden usar cambiar de Service (Exercise_sc2 con security on). Las signed pueden solicitarlo en el PRF de la forma siguiente:

```
<!ELEMENT servicesel EMPTY>
```

```
<!ATTLIST servicesel value (true|false) "true" >
```

Ejemplo: `<servicesel value="false"></servicesel>`

- **OJO:** Es posible que la selección de un Service implique que debe existir acceso al Return Channel si por ejemplo el nuevo Service necesita de este para bajarse la AIT (Interactive Channels)

Permission Request File: Inter-application communication policy

El Inter-App Comm API se ve en detalle en un capítulo específico: org.dvb.io.ixc.

Permission Request File: User Setting and Preferences access policy

- Las unsigned tendrán acceso de lectura únicamente a las siguientes preferencias:

User Language.

Parental Rating.

DefaultFontSize.

Country Code.

- Las signed pueden solicitar permiso de lectura y/o escritura a TODAS las preferencias de la forma:

```
<!ELEMENT userpreferences EMPTY>
```

```
<!ATTLIST userpreferences
```

```
write (true|false) "false"
```

```
read (true|false) "true" >
```

Ejemplo:

```
<userpreferences read="true" write="false"> </userpreferences>
```

El User Preferences API se ve en detalle en un capítulo específico: [org.dvb.user](#)

Permission Request File: Network Permissions

- Las unsigned apps NO tienen acceso a host:ports vía Return Channel
- Las signed apps pueden solicitar permiso mediante:

```
<!ELEMENT network (host)+>
```

```
<!ELEMENT host (#PCDATA)>
```

```
<!ATTLIST host action CDATA #REQUIRED >
```

Donde action puede ser: accept, connect. Ved JavaDoc de `java.net.SocketPermission`

Ejemplo:

```
<network>
```

```
<host action="connect">com.kke.cz:8893</host>
```

```
</network>
```

Permission Request File: Dripfeed

- Las unsigned apps NO tienen acceso a reproducir dripfeeds
- Las signed apps pueden solicitar permiso mediante el tag siguiente. Cuando lo tienen entonces podrán crear instancias de **DripFeedDataSource**

```
<!ELEMENT dripfeed EMPTY>
```

```
<!ATTLIST dripfeed value (true|false) "true" >
```

Ejemplo:

```
< dripfeed value="false"></ dripfeed >
```

Permission Request File: Runtime Code Extension Permission

- No se pueden crear ClassLoaders! Sólo se puede usar el org.dvb.lang.DVBClassLoader.
- Las Unsigned pueden cargar código a partir de “String Data” desde cualquier fuente (de ahí que podamos cargar código por http...)
- Signed apps SIN PRF pueden cargar código a partir de “String Data” desde cualquier fuente.
- Signed apps CON PRF **NO** pueden cargar código a partir de String Data salvo que se solicite poder usar String Data de procedencia interna o bien Interna/Externa en el PRF:

```
<!ELEMENT privilegedrce EMPTY>
```

```
<!ATTLIST privilegedrce
```

```
value (internal|external) "internal"
```

```
>
```

```
Ejemplo: <privilegedrce value="external"></privilegedrce>
```

Permission Request File: NON-CA smartcard

- Las unsigned NO tienen acceso
- Las signed pueden solicitarlo en el PRF mediante:
<!ELEMENT smartcardaccess EMPTY>

Ejemplo: <smartcardaccess/>

Permission Request File: Cryptographic Service Provider Management

- Las unsigned NO pueden añadir/quitar ningún proveedor.
- Las signed pueden solicitarlo mediante el uso del TAG siguiente, sin embargo los providers han de tener el mismo **organisation_id** que la app:

```
<!ELEMENT provider (name)+ >
```

```
<!ELEMENT name (#PCDATA)>
```

```
<!ATTLIST name scope (application|system) "application">
```

El Provider name debe ser: MHP_orgId.Provider_name.Card_name

Ejemplo:

```
<provider>
```

```
  <name scope="application">0x0000000B.EMV_PK11.VISA_REVOLVER</name>
```

```
</provider>
```

Permission Request File: Credenciales

- Una credencial sirve para dar acceso a un determinado recurso, en nuestro caso siempre será un fichero o un conjunto de ellos, nunca directorios.
- Para ello se deben de especificar todos los parámetros de seguridad oportunos. Ved el detalle del API en el capítulo 12 de las specs de MHP 1.1.2 (A0068r1.pdf).

```
<!ELEMENT persistentfilecredential (grantoridentifier,expirationdate,filename+,signature, certchainfileid)>
```

```
<!ELEMENT grantoridentifier EMPTY>
```

```
<!ATTLIST grantoridentifier id CDATA #REQUIRED >
```

```
<!ELEMENT expirationdate EMPTY>
```

```
<!ATTLIST expirationdate date CDATA #REQUIRED >
```

```
<!ELEMENT filename (#PCDATA)>
```

```
<!ATTLIST filename write (true|false) "true" read (true|false) "true" >
```

```
<!ELEMENT signature (#PCDATA)>
```

```
<!ELEMENT certchainfileid (#PCDATA)>
```

Permission Request File: Application Storage

- Las unsigned apps no tienen permiso para almacenar apps.
- Las signed apps pueden usar el API siempre que se solicite en el PRF de la forma:

```
<!ELEMENT applicationstorage (applicationstorageorg)*>
<!ATTLIST applicationstorage
manageservice: (true|false) "false"
createservice: (true|false) "false"
deleteservice: (true|false) "false"
managecache: (true|false) "false" >
<!ELEMENT applicationstorageorg EMPTY>
<!ATTLIST applicationstorageorg
orgid CDATA #REQUIRED
storeservice (true|false) "false"
removeservice (true|false) "false"
storecache (true|false) "false"
removecache (true|false) "false" >
```

El API de Stored Apps se ve en detalle en otro capítulo

¿ Cómo se Firma una app ?

- Para firmar una app vamos a manejar 3 tipos de ficheros:
 - **Hash Files**
 - **Signatures Files**
 - **Certificate Files**

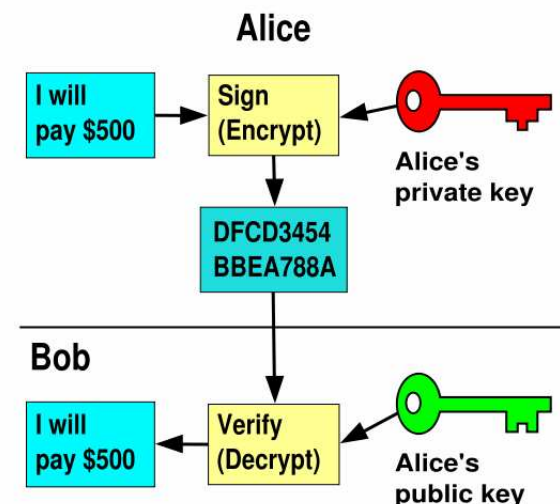
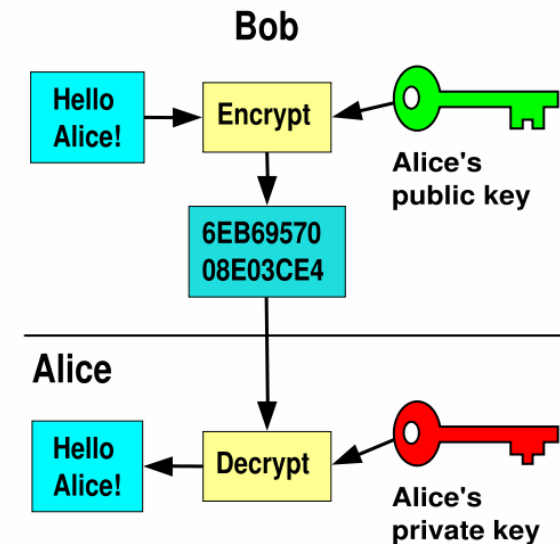
Veamos para qué sirven y cómo se generan cada uno de ellos.

- ¿ qué he de “firmar” de una app para decir **que está Firmada**, es decir: **que es signed** ?
 - En una aplicación firmada, todas las clases y ficheros de la misma que se carguen mediante el classpath deberán estar firmados por al menos el mismo conjunto de certificados por los que lo ha sido la clase XLET. Esto aplica por ejemplo, a todas las clases de la aplicación, a las imágenes o datos que se carguen usando `java.lang.Class.getResource()`.
 - Cuando se firma un JAR siguiendo el modelo de firma de MHP se considera que todo lo incluido dentro del jar está firmado

Un inciso. Un poco de Seguridad. Claves Públicas y Privadas

- Las Claves Públicas y Privadas se suelen generar partiendo de grandes números aleatorios. El algoritmo más común de encriptación es **RSA**.
- Decimos que **ENCRIPAMOS** cuando usamos nuestra **clave Pública** para codificar un mensaje que sólo se puede **desencriptar con la clave Privada correspondiente**.
 - Cualquiera puede **ENCRIPAR** usando la clave **PÚBLICA** pero sólo el que tiene la clave **PRIVADA** puede **DESENCRIPTAR** el mensaje.
 - La seguridad depende de lo bien guardada que esté la clave privada para que nadie pueda leer nuestros mensajes.**
 - Este esquema busca la Confidencialidad
- Decimos que **FIRMAMOS** un mensaje cuando usamos nuestra clave **Privada** para **encriptarlo**, de manera que sólo se puede desencriptar usando la clave **PÚBLICA**
 - Únicamente el que **ENCRIPTA** con la clave **PRIVADA** puede generar mensajes que puedan ser **DESENCRIPTADOS** por la clave **PUBLICA**. **Comprobamos el origen mediante la clave pública.**
 - La seguridad depende de lo bien guardada que esté la clave privada para que nadie suplante la identidad.**
 - Este esquema busca la Autenticidad

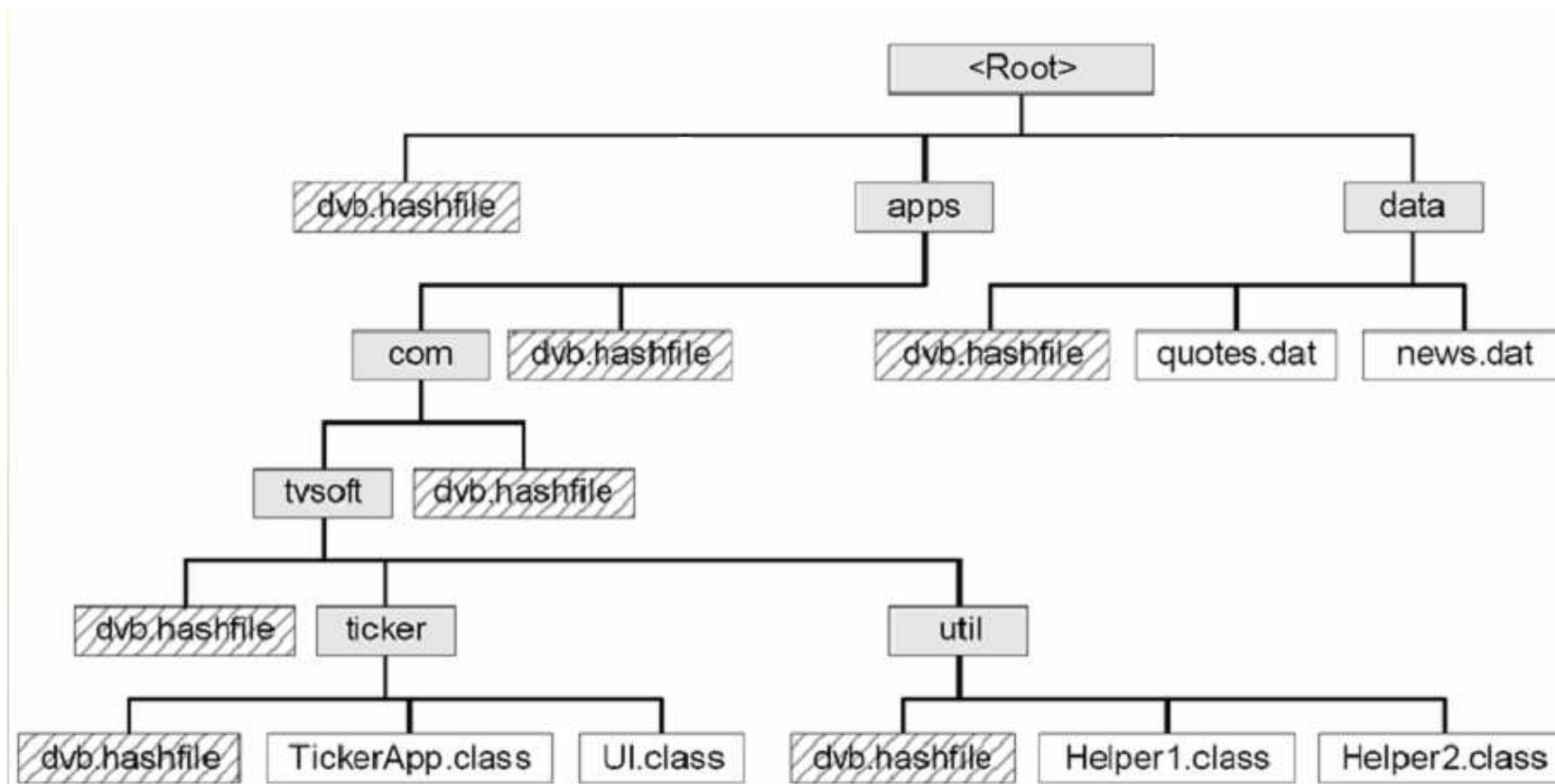
http://en.wikipedia.org/wiki/Public_key



¿ Cómo se Firma una app ? Paso 1: HASH Files

- La aplicación se compone de ficheros y directorios; imaginemos que nos encontramos en un directorio llamado **D1** que contiene **5 ficheros**. Para cada uno de estos ficheros generamos una cadena de caracteres que se llama **Digest**, de tal manera que incluimos los **5 Digest** dentro de un fichero llamado **dvb.hashfile** que se encuentra en el mismo directorio que los contiene. **El algoritmo para generar el Digest** puede ser MD5 o SHA-1.
- Mediante la comparación del Digest con el fichero sería posible saber si el fichero ha cambiado o no.
- En el directorio del que cuelga **D1** ocurre lo mismo: existen una serie de ficheros y nuestro directorio **D1**. Pues bien, en este también existirá un fichero **dvb.hashfile** que contendrá los **Digest** de todos los ficheros que contiene y también el Digest obtenido del fichero **dvb.hashfile** del directorio **D1**. Por su puesto si hubiera más directorios incluiríamos los Digests generados a partir de sus dvb.hashfile files...
- Y así sucesivamente hasta llegar al directorio raíz, donde tendremos el **dvb.hashfile** de mayor nivel.

¿ Cómo se Firma una app ? Paso 1: HASH Files (y 2)



MHP 1.1.2, A0068r1

¿ Cómo se Firma una app ? Paso 1: HASH Files (y 3)

- Los Hash files vienen a contener la lista de todos los componentes del directorio donde se encuentran con su respectivo Digest, **o NO**: Es posible no tener que “codificar” absolutamente todo, podemos por ejemplo dejar fuera de la encriptación ficheros de datos o imágenes.
- En cualquier caso **todos los elementos deben estar referenciados dentro del Hashfile**. (Ved 12.4.1 de MHP specs A0068r1)
- También se pueden generar los Digest para grupos de Ficheros, en lugar de uno por fichero, sin embargo esta práctica no es muy recomendable ya que aunque por ejemplo sólo necesitemos acceder a uno de ellos este esquema obliga a abrir y comprobar TODOS.

¿ Cómo se Firma una app ? Paso 1: HASH Files (y 4)

- Formato del fichero HASH. Como véis name_count nos indica que un Digest puede agrupar a varios ficheros.

Table 120: Syntax of the Hashfile

Syntax	Num. Bits	Format
Hashfile () {		
digest_count	16	uimsbf
for(i=0; i<digest_count; i++) {		
digest_type	8	uimsbf
name_count	16	uimsbf
for(j=0; j<name_count; j++) {		
name_length	8	uimsbf
for(k=0; k<name_length; k++) {		
name_byte	8	bslbf
}		
}		
for(j=0; j<digest_length; j++) {		
digest_byte	8	bslbf
}		
}		
}		
Other data may follow but can be ignored by implementations conforming to this profile.		

¿ Cómo se Firma una app ? Paso 1: HASH Files (y 5)

- Valores de digest_type

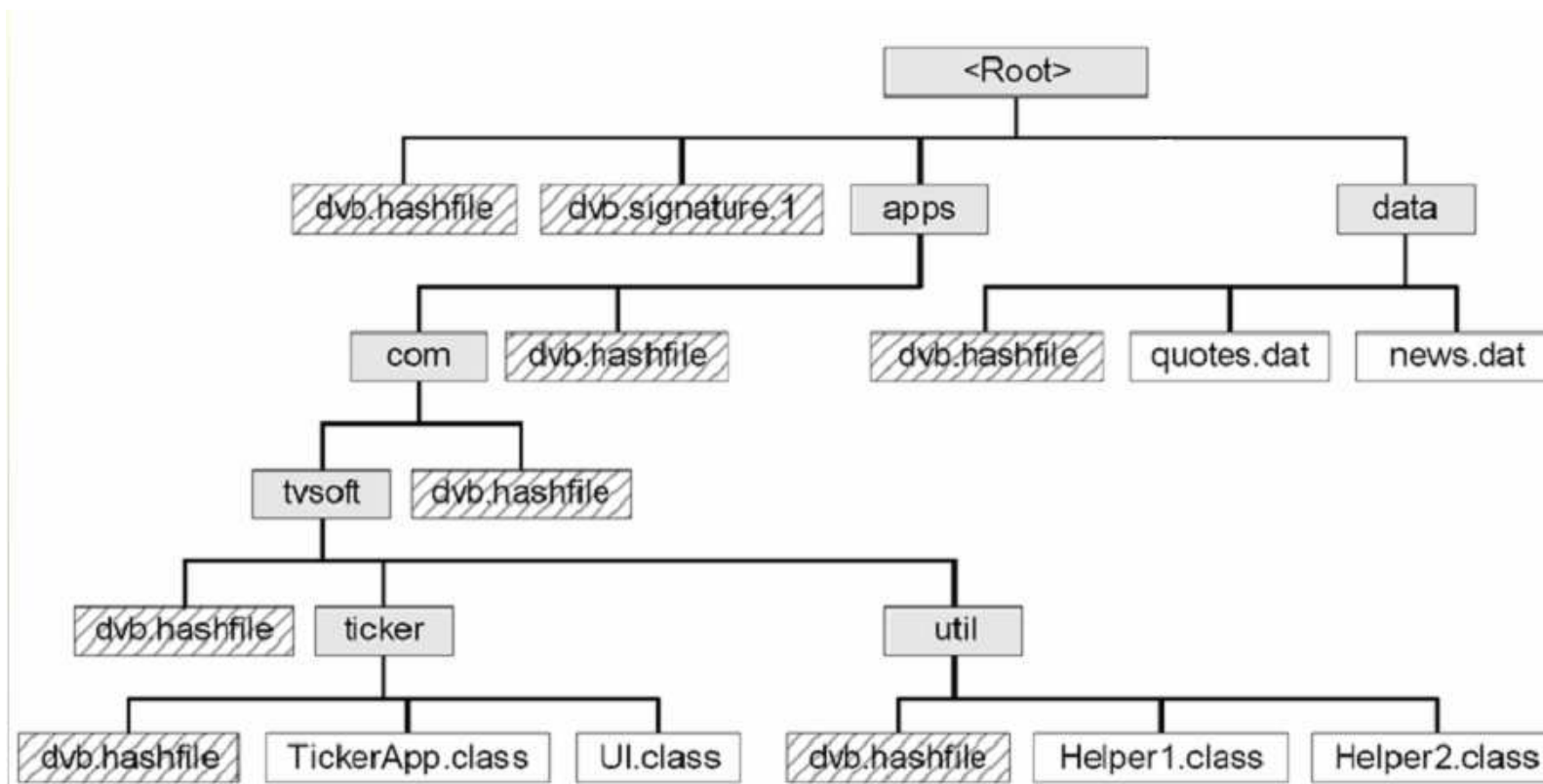
Table 121: Values of digest_type

value	digest len.	algorithm
0	0	Non authenticated
1	16	Digest computation rules without prefix and with MD-5 as defined in RFC 1321 [38]
2	20	Digest computation rules without prefix and with SHA-1 as defined in FIPS-180-1 [62]
3	20	Digest computation rules with prefix and with SHA-1 as defined in FIPS-180-1 [62] .
Other values		Reserved for future use

¿ Cómo se Firma una app ? Paso 2: Signature Files

- Disponer únicamente de Hash files no es muy seguro, pues simplemente regenerándolos después de haber sustituido, por ejemplo una clase, nos permitiría “engañar” al sistema. Hemos de hacer algo más.
- Cuando hemos llegado al directorio raíz tenemos finalmente un **dvb.hashfile** que en realidad es como un Digest de toda la APP, pues lo hemos ido construyendo de abajo a arriba. **Ha llegado el momento de Firmar la APP.**
- Para firmar la APP generamos el **Digest** del **dvb.hashfile** raíz y este lo **Firmamos**, es decir, el **Network Operator** lo encripta usando su **clave Privada** aplicando el algoritmo **RSA**, y guarda el resultado en el fichero **dvb.signature.id**, donde id empieza en 1 y sirve para identificar varios posibles “Firmantes”. Este fichero **dvb.signature.id** reside en el mismo directorio que el **dvb.hashfile** raíz.

¿ Cómo se Firma una app ? Paso 2: Signature Files (y 2)

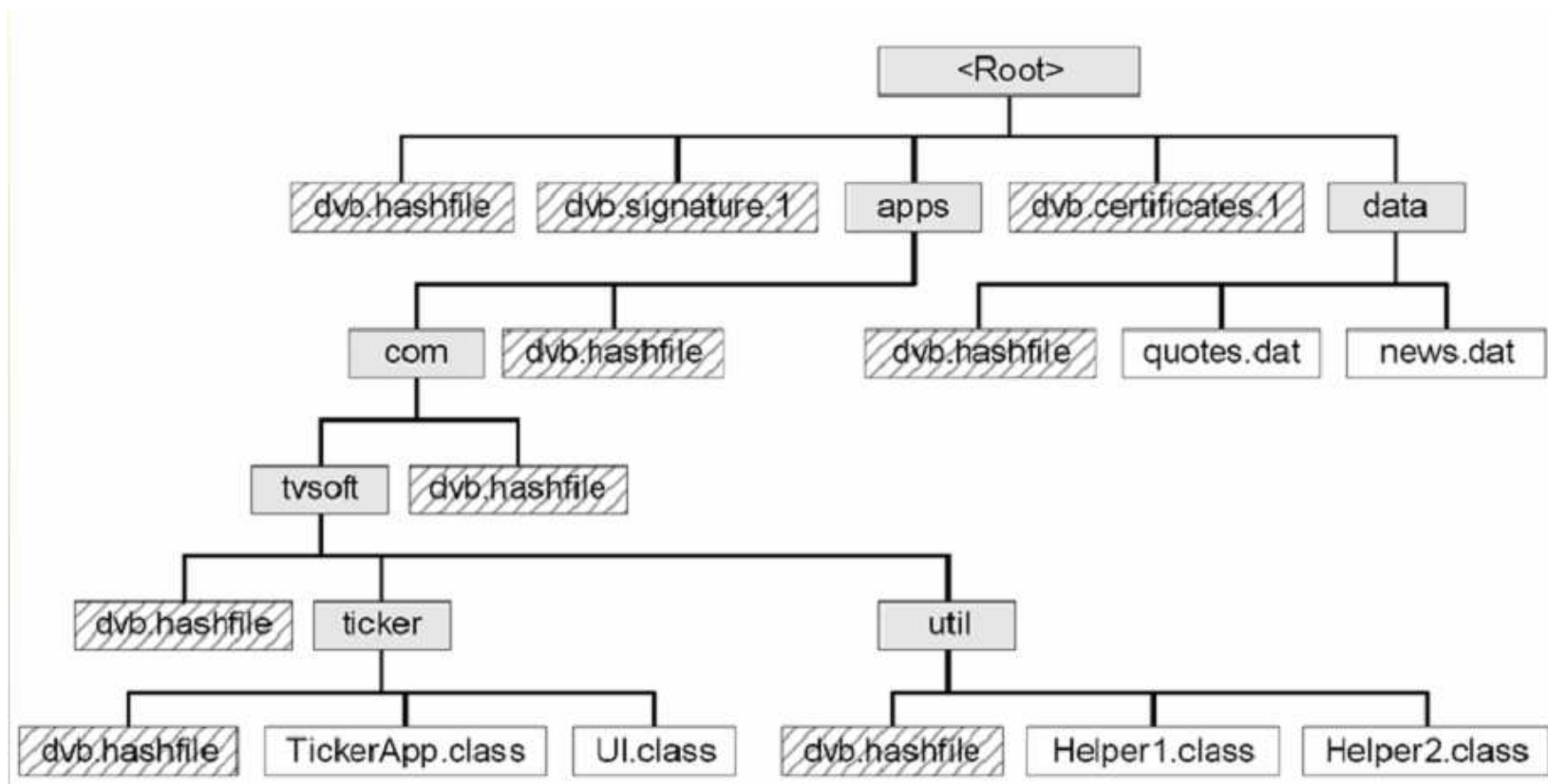


MHP 1.1.2, A0068r1

¿ Cómo se Firma una app ? Paso 3: Certificate Files

- Así pues hemos emitido la app, con sus ficheros hashed, con el hash padre a su vez hashed y además este resultado lo hemos **Firmado** por una **clave privada** que sólo se puede abrir con la **Pública**, ¿ pero donde está la **clave pública correspondiente a la privada que se usó para firmar el Digest del último Hash ?**
- Se incluirá un **CERTIFICADO** en el directorio raíz con dicha clave pública, un **certificado que me asegura que dicha clave pertenece al Network Operator: el “Firmante”**. Este certificado reside en un fichero en el directorio raíz que se llamará **dvb.certificate.id**, donde id empieza en 1 y tiene el mismo valor que el **dvb.signature.id** correspondiente.
- El tipo de certificado que se usa es una variante de Internet Profile de **X.509** definida en la RFC 2459.

¿ Cómo se Firma una app ? Paso 3: Certificate Files (y 2)



MHP 1.1.2, A0068r1

¿ Cómo se Firma una app ?

- Un certificado puede estar a su vez “certificado” por otra CA generando lo que se llama una **certificates chain** hasta que llegamos a un certificado emitido por una **Entidad Reconocida** en cuyo caso el certificado se denomina **Root Certificate**. Todo esto se almacena dentro del mismo fichero de Certificado:

Mhp 1.1.2 A068r1

12.4.3 CertificateFile

12.4.3.1 Description

The CertificateFile contains all of the certificates in the certificate chain up to, **and including, the root certificate**. The leaf certificate is placed first in the file. The last certificate in the file is the root certificate.

- Para poder autenticar la app con un certificado determinado el Root Certificate contenido en el certificado debe ser reconocido por el STB, y el modo en que lo haga depende de la plataforma. MHP intentará el proceso con todos los certificados/signatures que encuentre.

¿ Cómo se Firma una app ?

- Dentro del STB el fabricante incluirá un Root Certificate. **Ved www.dvbservices.org**. Texto interesante de la web:

The DVB MHP PKI provides the backbone for authenticating applications and files that are broadcast to an MHP receiver. This authentication process is designed to ensure that application code has not been modified from that originally developed by its author, is permitted to execute on devices connected to the broadcast network, and that the application only accesses the set of restricted resources that it has been authorised to access.

The DVB MHP PKI relies on the integrity of certificates that verify the identity of the entity who has signed an application's file tree. **The highest level certificate, known as the "root certificate" is embedded into each MHP receiver by its manufacturer**, and this is used to authenticate a certificate at the next level in the hierarchy. This authentication process is repeated through each level of the certificate hierarchy until a leaf (or end-entity) certificate is encountered and authenticated. Thus users can be sure that application has not been modified and is only using the intended resources.

¿ Cómo se Firma una app ?

- Para poder generar signed APPs hay que seguir el proceso definido en la web indicada antes y así obtener el Certificado y la clave privada para poder generar las Signatures...
- Bajados de dicha web:
 - DVB MHP Subscriber Private Key Protection Guide [subprivkeyguide.pdf](#)
 - DVB MHP Subscriber Support Guide [subsupportguide.pdf](#)

¿ Cómo se Firma una app ?

- El proceso de validación de una aplicación firmada es muy arduo y por lo tanto el STB debe de estar seguro de que la app lo está antes de iniciar el proceso.
- Para indicarle al STB que una app está firmada o no usaremos el appid de la siguiente forma:
 - 0x0000 - 0x3fff: serán usados como ID de aplicaciones NO Firmadas
 - 0x4000 - 0x7fff: serán usados como ID de aplicaciones Firmadas

¿ cómo sé si un fichero está firmado o no ?

- Llamando al método siguiente de `org.dvb.dsmcc.DSMCCObject` obtendremos los Certificados usados para sus firmas.

```
public X509Certificate[][] getSigners()
```

¿ Cómo se Firma una app ? Actualización y revocación de Certs.

- Los Root Certificates pueden revocarse o Actualizarse mediante mensajes del tipo Root Certificate Management Messages, RCMMs, que se mandan en el Broadcast.
- Igualmente los certificados almacenados en el STB se pueden revocar. Esto se hace transmitiendo igualmente el Broadcaster una CRL: Certificate Revocation List que contiene los serial de los Certificados que deben ser revocados.

Recomendación

- Con el fin de mejorar la eficiencia de la ejecución, y dado que el proceso de validar certificados exige recursos conviene no cargar por el ClassPath aquellos ficheros que no son estrictamente necesarios, y podemos usar: `java.io.File` o `DSMCCObject`....

ISO/IEC 13818-1	Part 1. Elementary Streams transport definition
ISO/IEC 13818-6	Part 6. Extensions for DSM-CC. Digital Storage Media Command and Control
ETSI EN 300 468	Digital Video Broadcasting (DVB);Specification for Service Information (SI) in DVB systems
ETSI EN 301 192	DVB specification for data broadcasting
ETSI TR 101 202	Implementation Guidelines for Data broadcasting
ETSI TR 101 162	Digital broadcasting systems for television, sound and data services; Allocation of Service Information (SI) codes for Digital Video Broadcasting (DVB) systems
ETSI TR 102 154	Implementation guidelines for the use of MPEG-2 Systems, Video and Audio in Contribution and Primary Dist
ETSI TR 101 211	Guidelines on implementation and usage of Service Information (SI)
ETSI TR 101 200	Digital Video Broadcasting (DVB); A guideline for the use of DVB specifications and standards
DAVIC	Digital Audio Visual Council. davic 1.4.1
HAVI	Specification of the Home Audio/Video Interoperability (HAVi) Architecture
Interactivetvweb	http://www.interactivetvweb.org/
Wikipedia DSMCC	http://en.wikipedia.org/wiki/DSM-CC
MHP 1.1.2	Multimedia Home Platform, A068r1 & tam668r23_11xdraft_20061115
MHP 1.1.3	Multimedia Home Platform, A068r3
CDC 1.1	Connected Device Configuration (CDC) 1.1 (JSR=218).
PBP 1.1	Personal Basis Profile 1.1 (JSR 217)
MHP.org	www.mhp.org
INTRO MHP 1.1.3	tam1032r1-mhp-iptv-presentation