



MANUAL DE USUARIO

DNIEdroid v2.2

4 de marzo de 2019

DNIEDROID v2.2

Versión	Fecha	Autores	Descripción
1.0	19.02.2018	CNP-FNMT	Creado
1.1	20.02.2018	CNP-FNMT	Versionado y corrección de elementos gráficos.
1.2	14.12.2018	CNP-FNMT	Actualización de funcionalidad.
1.3	01.02.2019	CNP-FNMT	Actualización de funcionalidad.
1.4	28.02.2019	CNP-FNMT	Migración de cliente HTTP y reorganización de paquetes.

Índice

1. Introducción	5
2. Objetivo	5
3. Arquitectura	6
3.1. Lógica	6
3.2. Física	7
4. Integración en aplicaciones Android	8
4.1. Introducción	8
4.2. Requisitos	8
4.3. Funcionalidad	9
4.4. Paquetes disponibles	9
4.5. Comprobación de edad	28
4.6. Obtener estado del certificado	29
4.7. Información adicional	30

Índice de figuras

1.	Reverso DNLe v3.0	5
2.	Arquitectura lógica DNLeDroid	7
3.	Integración en Android	8
4.	Acceso a servicios	9

Las comunicaciones entre el dispositivo y el DNIE se realizarán siempre cifradas según la norma [4].

En este documento nos centraremos en la versión inalámbrica del DNIE, en la que la interacción desde el punto de vista físico se realiza por medio de NFC.

Para información relativa a los certificados X509 incluidos en el DNIE 3.0, véase [1]. En cuanto a la conexión por radiofrecuencia entre dispositivos, ver [2].

3. Arquitectura

3.1. Lógica

El DNIEDroid es un middleware encargado de gestionar la conexión entre dispositivos móviles y el DNIE. Ofrece un API básico de operaciones que permite a los desarrolladores gestionar de manera transparente las conexiones NFC con el DNIE.

De esta manera se pretende facilitar la implementación de aplicaciones, ya que no hará falta conocer en detalle el funcionamiento de la tarjeta sino que bastará con incluir el DNIEDroid en la aplicación para poder acceder a las funcionalidades del DNIE.

En el diagrama 2 se describe la arquitectura lógica actual del componente, desarrollado en Java para Android.

DNIEDroid ofrece una capa de alto nivel para la gestión de la conexión con dispositivos lectores de tarjetas. Esta capa, se apoya en una representación de “alto nivel” de los dispositivos, gestionando el envío y recepción de comandos a través del canal de comunicación (USB o NFC). Esta gestión se lleva a cabo gracias a la capa de abstracción de bajo nivel proporcionada por Android, y que en su caso debería proporcionar también en sistemas como iOS, Windows Phone, etc.

El objetivo del componente DNIEDroid es interactuar con el DNIE en dispositivos basados en Java (como smartphones y tablets Android), ya sea mediante interfaz USB o mediante NFC. De esta manera se facilita el acceso desde esas plataformas a los servicios que hacen uso del DNIE para autenticación y firma electrónica.

En el diagrama 3 se describe el modo de integración del DNIEDroid en DD4J.

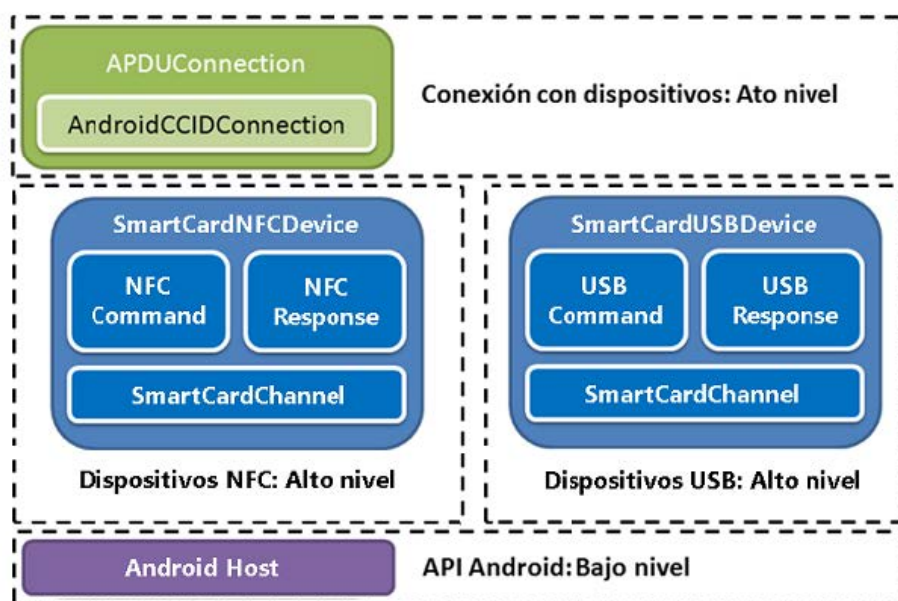


Figura 2: Arquitectura lógica DNIEDroid

Tal como se observa en el diagrama, el componente DNIEDroid se integra como una conexión subyacente más dentro de la arquitectura del Controlador Java para el DNIE.

Se abstrae así a las capas superiores de la interacción con los lectores de tarjetas conectados a los dispositivos Android (ya sea vía USB o NFC) y la gestión de intercambio de comandos de bajo nivel. Es esta última capa la que da acceso al API de NFC en Android y que nos permite el envío de comandos y respuestas entre DNIE y dispositivo.

3.2. Física

A pesar de que el componente es puramente lógico, está diseñado para la comunicación de dispositivos Android mediante dos posibles interfaces.

En primer lugar con lectores de tarjetas USB conectados a él mediante un adaptador USB-OTG (On The Go), ya que la mayoría de dispositivos Android disponen de un puerto micro USB en lugar de USB y no hay, a día de hoy, lectores de SmartCard micro USB. En caso de que el dispositivo disponga de un puerto USB, no haría falta dicho adaptador.

En segundo lugar mediante la conexión por proximidad con un DNIE v3.0 mediante tecnología NFC. Ésta permite la comunicación entre el dispositivo Android y el DNIE sin necesidad de lector de tarjetas.

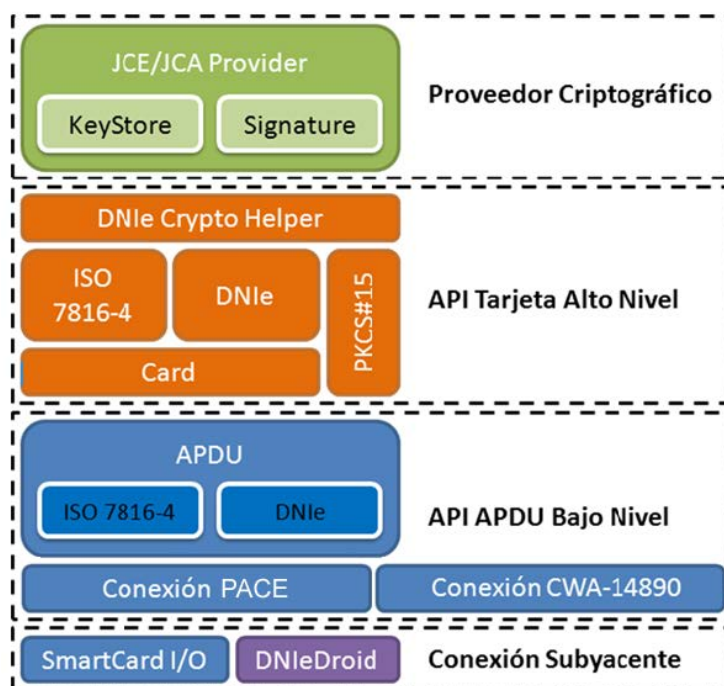


Figura 3: Integración en Android

4. Integración en aplicaciones Android

4.1. Introducción

Tras la integración del controlador DNleDroid en la aplicación Java, es posible acceder a toda la funcionalidad proporcionada por éste desde aplicaciones desarrolladas para dispositivos Android.

Este apartado pretende servir de guía a la hora de la integración del controlador, adaptado para Android en aplicaciones de terceros.

4.2. Requisitos

Para poder realizar un uso adecuado de la librería es necesario tener experiencia en desarrollo de aplicaciones para dispositivos Android, y se recomienda poseer conocimientos básicos sobre Infraestructura de Clave Pública (PKI) y de Java Cryptography Architecture (JCA), así como de protocolos de comunicación a través de las redes.

Esta librería se apoya en APIs de terceros, por lo que será necesario incluir dependencias en el proyecto. Así, en el archivo *build.gradle* se añadirán las

4 INTEGRACIÓN EN APLICACIONES ANDROID DNIEDROID v2.2

siguientes líneas:

```
implementation 'org.bouncycastle:bcprov-jdk15on:1.49'  
implementation 'org.jsoup:jsoup:1.10.3'
```

4.3. Funcionalidad

Como funcionalidad básica, la librería dispone de la implementación de la capa criptográfica accesible a través de un Provider, véase [6], proporcionando una interfaz estándar.

También se incluyen clases para las comunicaciones y recuperación de información a través de la red, que abstrae al usuario de la complejidad en la implementación de la autenticación con certificado de cliente a través del protocolo HTTPS.



Figura 4: Acceso a servicios

Además, la librería facilita un rápido desarrollo de la aplicación a través de la incorporación de clases que extienden de [android.app.Fragment](#), estableciendo las conexiones NFC necesarias y proporcionando un entorno gráfico por defecto, para ser integradas en el [android.app.Activity](#).

4.4. Paquetes disponibles

A continuación se indica una breve descripción de los paquetes que engloban toda la funcionalidad necesaria para realizar la integración de la aplicación. La mayor parte del código fuente que se muestra en los ejemplos se ha extraído del proyecto que incluye el paquete de desarrollo.

es.gob.jmulticard.jse.provider contiene las clases que implementan la capa criptográfica, disponible a través de la interfaz JCA.

es.gob.fnmt.nfc contiene las clases para la comunicación con la tarjeta criptográfica a través de NFC.

es.gob.fnmt.net contiene las clases para la conexión y recuperación de información a través de la red.

es.gob.fnmt.gui contiene las clases relacionadas con la interfaz de presentación e interacción para el usuario.

es.gob.fnmt.policy contiene las clases para el uso de claves de la tarjeta criptográfica a través de políticas.

es.gob.jmulticard.ui.passwordcallback contiene la interfaz necesaria para la implementación del diálogo de petición de PIN.

de.tsenger.androsmex.mrtd contiene las clases que encapsulan la información de tipo MRTD, véase [5].

de.tsenger.androsmex.data contiene las clases que encapsulan la información del CAN (Card Access Number),

Paquete *es.gob.jmulticard.jse.provider*

DnieProvider: clase que implementa la interfaz *java.security.Provider* como punto de acceso a toda la capa criptográfica.

DnieKeyStore: clase que permite directamente instanciar un objeto de tipo *java.security.KeyStore*, que sirve para dar acceso a la información MRTD. Como argumentos se pasan: el provider y una instancia de la implementación de la especificación de *java.security.KeyStoreSpi* para MRTD, que requiere indicar el tag NFC obtenido en la detección del dispositivo a través de la interfaz [android.nfc.NfcAdapter.ReaderCallback](#), y el código de 6 dígitos que corresponde al CAN del DNIe, véase [7]

A continuación se indica un código de ejemplo básico, disponible en *SampleActivity-provider*.

4 INTEGRACIÓN EN APLICACIONES ANDROID DNIEDROID v2.2

```
1    ...
2    String can;
3    Tag tag;
4    //... (una vez recuperamos Tag y CAN)
5    final DnieProvider p = new DnieProvider();
6    Security.insertProviderAt(p, 1);
7    KeyStore keyStore = new DnieKeyStore(new MrtdKeyStoreImpl(can, tag), p);
8    EF_COM data = ((DnieKeyStore)keyStore).getEF_COM();
9    ...
```

Paquete *es.gob.fnmt.nfc*

NFCCommReaderFragment: clase que hereda de *android.app.Fragment* y que implementa la interfaz *android.nfc.NfcAdapter.ReaderCallback*, proporcionando notificación de eventos a través de la interfaz *NFCCommReaderFragmentListener*, que incluye la propia clase. Implementando esta interfaz, se recibirán los eventos relacionados con la comunicación por NFC con el DNIE: *getCanToStore()* y *doNotify()*.

getCanToStore() espera devolver un objeto de la clase *de.tsenger.androsmex.data.CANSpecDO* para ser almacenado/actualizado en el almacén interno de la aplicación. Así, en el próximo inicio de la aplicación, se podrán listar los CAN disponibles. En caso de retornar el objeto, éste sustituirá al que se pasó en la inicialización del *Fragment*. Si se retorna *null*, no se actualizará.

doNotify() informa de los eventos durante la lectura del dispositivo NFC del DNIE v3.0. Es útil, por ejemplo, para actualizar la interfaz gráfica (detección del DNIE, etc.).

El siguiente código es un ejemplo de instanciación de un objeto e inicio de transición al *Fragment*, donde se observa que es necesario pasar como argumento un objeto *de.tsenger.androsmex.data.CANSpecDO* en el que se indica el CAN. Opcionalmente, se puede indicar también si se requiere una precarga de claves (por defecto 'false'), que implica la solicitud del PIN. Ejemplo en *SampleActivity_api*.

```
1    ...
2    CANSpecDO can = new CANSpecDO("123456", "", "");
3    Bundle arg = new Bundle();
```

DNIEDROID v2.2 4 INTEGRACIÓN EN APLICACIONES ANDROID

```
4  //CAN
5  arg.putParcelable(NFCCommReaderFragment.CAN_ARGUMENT_KEY_STRING, can);
6  //precarga de claves
7  arg.putBoolean(NFCCommReaderFragment.PRELOADKEYSTORE_ARGUMENT_KEY_STRING
8  ↪ , true);
9  NFCCommReaderFragment readerFragment = new NFCCommReaderFragment();
10 readerFragment.setArguments(arg);
11 FragmentTransaction transaction =
12 ↪ getFragmentManager().beginTransaction();
13 transaction.replace(R.id.fragment_container, readerFragment);
14 transaction.addToBackStack(null);
15 transaction.commit();
16 ...
```

Internamente, esta clase inicializa el *DnieProvider* y el *DnieKeyStore* cuando establece comunicación con el DNIe, pudiéndose recuperar el almacén de claves (implementado siguiendo JCA) mediante el método estático *NFCCommReaderFragment.getKeyStore()*. En el siguiente código se muestran ejemplos de implementación de los métodos de la interfaz *NFCCommReaderFragment.NFCCommReaderFragmentListener*.

Recepción de la notificación del evento de solicitud de identificación de CAN para persistencia de datos. Una vez devuelto, este objeto tipo *CANSpecDO* se almacenará y podrá recuperarse en un nuevo acceso a la aplicación. Ejemplo en *SampleActivity_api*.

```
1  @Override
2  public CANSpecDO getCanToStore(KeyStore keyStore, String can) throws
3  ↪ KeyStoreException {
4      X509Certificate certificate = (X509Certificate)
5      ↪ keyStore.getCertificate(DnieKeyStore.SIGN_CERT_ALIAS);
6      return new CANSpecDO(can, Tool.getCN(certificate),
7      ↪ Tool.getNIF(certificate));
8  }
```

Recepción de la notificación de los eventos de comunicación por NFC con el DNIe. Ejemplo en *SampleActivity_gui*.

```
1  @Override
2  public void doNotify(final NFC_callback_notify notify, String msg) {
```

4 INTEGRACIÓN EN APLICACIONES ANDROID DNIEDROID v2.2

```
3      String message =msg;
4      switch (notify){
5          case NFC_TASK_INIT:
6              _readerFragment.updateInfo(notify, "Inicio comunicación DNIE...",
              ↳ message);
7              break;
8          case NFC_TASK_UPDATE:
9              _readerFragment.updateInfo(notify, "Leyendo DNIE...", message);
10             break;
11          case ERROR:
12              _readerFragment.updateInfo(notify, "Error en comunicación",
              ↳ message);
13              if(msg.contains("CAN incorrecto")){
14                  Toast.makeText(this.getApplicationContext(),msg,
              ↳ Toast.LENGTH_LONG).show();
15              }
16              return;
17          }
18          break;
19          case NFC_TASK_FINISHED:
20              //Lanzamos la tarea de firma dentro de un proceso asíncrono. Si
              ↳ no, los diálogos de verificación de firma no aparecerán.
21              new TaskSignature().execute();
22              break;
23          default:
24              _readerFragment.updateInfo(notify, "Comunicando con Dnie",
              ↳ message);
25      }
```

De forma opcional se puede implementar otra interfaz, *NFCCommReaderFragment.FinalTaskListener*, que permite la ejecución de una tarea final una vez haya finalizado correctamente la carga de claves con el DNIE. Se registra la clase implementadora con *NFCCommReaderFragment.setFinalTask()*, que permite invocar los métodos de forma automática, siendo éstos *doFinalTask()* y *finalTaskDone()*.

doFinalTask() ejecuta el método de la interfaz implementado por la clase, que realiza las acciones deseadas.

finalTaskDone() ejecuta el método de finalización de tarea, en el caso de que no se haya producido ningún error previo.

Ejemplo en *SampleActivity_api*.

DNIEDROID v2.2 4 INTEGRACIÓN EN APLICACIONES ANDROID

```
1  ...
2  NFCCommReaderFragment readerFragment = new NFCCommReaderFragment();
3  readerFragment.setFinalTask(this); //Registramos la clase que
   ↳ implementa la interfaz para realizar la tarea final.
4  ...
5  @Override
6  public void doFinalTask(){
7      try {
8          updateInfo("Realizando firma...", null);
9          Signature signatureEngine = Signature.getInstance("SHA256withRSA",
   ↳ new DnieProvider());
10         signatureEngine.initSign((PrivateKey)
   ↳ _keyStore.getKey(DnieKeyStore.SIGN_CERT_ALIAS, null));
11         signatureEngine.update(Tool.EXAMPLE_TEXT.getBytes());
12         signature = signatureEngine.sign();
13     }
14     catch (Exception e){
15         e.printStackTrace();
16     }
17 }
18
19 @Override
20 public void finalTaskDone() {
21     updateInfo("Firma realizada.", Base64.encodeToString(signature,
   ↳ Base64.DEFAULT));
22 }
23 ...
```

Paquete *es.gob.fnmt.net*

Este paquete a su vez se organiza en otros dependiendo de la funcionalidad que aporta: seguridad (*.ssl*), conectividad (*.http*) o herramientas (*.tool*).

Paquete *es.gob.fnmt.net.ssl*

DNIESSLSocketFactory: clase que permite instanciar la factoría de sockets *javax.net.ssl.SSLSocketFactory* necesaria para establecer conexiones por protocolo HTTPS, implementando la autenticación de cliente con certificado digital mediante el DNIE. Utilizará el *java.security.KeyStore* de tipo *es.gob.jmulticard.jse.provider.DnieKeyStore* para resolver las firmas di-

4 INTEGRACIÓN EN APLICACIONES ANDROID DNIEDROID v2.2

gitaes necesarias en el protocolo *SSL/TLS handshake*. Ejemplo en *SampleActivity-connect_gui*.

```

1  ...
2  SSLSocketFactory dniesSLSSocketFactory = DNIESSLSocketFactory.getInstance()
   ↳ (NFCCommReaderFragment.getKeyStore(), //almacén del DNIE con los
   ↳ certificados de usuario.
3  Toolbox.getKeyStoreFromResource(R.raw.truststore,
   ↳ getApplicationContext()), //almacén con los certificados de CA
   ↳ reconocidos para la conexión con los servidores.
4  new KeyManagerPolicy.Builder().init().addAlias(DnieKeyStore.AUTH_CERT_)
   ↳ ALIAS).build(), //política de selección de certificados de
   ↳ usuario. En este caso indicamos el certificado de autenticación.
5  getApplicationContext();
6  ...

```

A través de este constructor se pueden indicar, además de los almacenes de certificados de cliente y de certificados de CA de confianza visto en el código fuente anterior, las políticas de selección de certificados y el protocolo seguro de comunicaciones (SSLv3, TLSv1.1, etc.).

Paquete *es.gob.fnmt.net.http*

HTTPClient: clase que encapsula la funcionalidad de un cliente sencillo para peticiones HTTP de tipo ‘GET’ y ‘POST’, a través de la API *javax.net.ssl.HttpsURLConnection*. Permite opcionalmente el paso de parámetros, establecer la factoría de sockets SSL/TLS para el establecimiento del canal seguro, así como indicar propiedades específicas requeridas para poder realizar la conexión, como son las cabeceras HTTP. Ejemplo en *SampleActivity-connect_gui*.

```

1  HTMLParser parser = new HTMLParser(getApplicationContext()); //clase de
   ↳ ayuda para obtener información de las páginas HTML.
2  ...
3  parser.setContent(new HTTPClient(URL_SS_LOGIN).getStringResponse());
4  Element dniesElement = parser.getDocument().getElementById("loginCertific_
   ↳ adoFormSubmit").parent().getElementsByTag("input").first();
5  String finalUrl = null;
6

```

DNIEDROID v2.2 4 INTEGRACIÓN EN APLICACIONES ANDROID

```
7 postParams.put("loginCertificadoFormSubmit",
  ↳ dnieElement.getElementsByAttributeValue("name",
  ↳ "loginCertificadoFormSubmit").first().val());
8 finalUrl = URL_TUSS_DOMAIN + dnieElement.parent().attr("action");
9 updateProgressDlg(null, "Llamada "+ ++llamada);
10
11 parser.setContent(new HTTPClient(finalUrl,
  ↳ postParams).getStringResponse(dnieSSLConnectionFactory));
12 updateProgressDlg(null, "Llamada "+ ++llamada);
13 ...
```

Para establecer la conexión y recuperar los datos, se llamará a uno de los métodos disponibles, dependiendo del formato requerido.

```
1 int httpCode = new HTTPClient(URL_SS_LOGIN).getResponse(); //Simplemente
  ↳ obtenemos el código HTTP de respuesta (p.ej. 200);
2 InputStream is = new HTTPClient(URL_SS_LOGIN).getStreamResponse();
  ↳ //Devuelve el flujo de datos.
3 String html = new HTTPClient(URL_SS_LOGIN).getStringResponse();
  ↳ //Devuelve una cadena.
4 byte[] data = new HTTPClient(URL_SS_LOGIN).getByteArrayResponse();
  ↳ //Devuelve una array de bytes.
```

Por defecto las peticiones se realizan a través del método ‘GET’ y, en el caso de recuperar caracteres, con la codificación *UTF-8*. Sin embargo, las funciones sobrecargadas permiten establecer estos parámetros según sea necesario.

Por último, una vez realizada la conexión y obtenidos los datos, se puede recuperar el código HTTP devuelto para verificar que la conexión ha sido exitosa, o realizar alguna acción en función de su valor (4XX, 5XX, etc).

```
1 HTTPClient httpClient = new HTTPClient(URL_SS_LOGIN);
2 InputStream is = httpClient.getStreamResponse();
3 int httpCode = httpClient.getLastResponse(); //Recuperamos el código
  ↳ devuelto por la conexión.
```

CookiePolicyHandler: clase que permite gestionar las cookies de los diferentes sitios que se van visitando. Para almacenar las cookies y así dar

4 INTEGRACIÓN EN APLICACIONES ANDROID DNIEDROID v2.2

persistencia a la autenticación realizada en los diferentes dominios, es necesario registrar un gestor de cookies *java.net.CookieManager* a través de *java.net.CookieHandler*. Con un objeto *CookiePolicyHandler* asociado al *CookieManager*, tendremos acceso para poder eliminar las cookies de un sitio en concreto cuando sea necesario. Ejemplo en *SampleActivity_main* y *SampleActivity_connect_gui*.

```
1    ...
2    public static CookieManager _cookieManager = new CookieManager();
3    public static CookiePolicyHandler _cookiePolicy = new
    ↪ CookiePolicyHandler();
4
5    static{
6        _cookieManager.setCookiePolicy(_cookiePolicy);
7        CookieHandler.setDefault(_cookieManager);
8    }
9    ...
10   //Establecemos el dominio para tener un control de las cookies que se
    ↪ van guardando.
11   SampleActivity_main._cookiePolicy.setSiteHandling("TUSS");
12   ...
13   //Eliminamos las cookies que pertenecen a este dominio.
14   SampleActivity_main._cookiePolicy.deleteCookies(SampleActivity_main._coo
    ↪ kieManager.getCookieStore());
15   ...
```

MultipartData: clase que facilita el envío de archivos en las peticiones HTTP, a través de una interfaz sencilla para la construcción del contenido. Se muestra un ejemplo en el siguiente código:

```
1    ...
2    //Request
3    MultipartData multipartData = new MultipartData();
4    multipartData.addPart("fileData", file, context); //Archivo a enviar.
5    multipartData.addPart("codigoSeguridad", code); //Parámetro requerido.
6    HTTPClient httpClient = new HTTPClient(VÁLIDE_FIRMA, multipartData);
    ↪ //Cliente HTTP al que le pasamos el objeto.
7    String response = httpClient.getStringResponse();
8    if (httpClient.getLastResponse() != HttpURLConnection.HTTP_OK)
9        throw new ConnectException("Cannot send the file to Valide");
10    ...
```

Paquete *es.gob.fnmt.net.tool*

HTMLParser: clase que encapsula la funcionalidad de un simple ‘parser’ de documentos con sintaxis HTML. Internamente hace uso del paquete *org.jsoup*. Se puede ver su uso en los varios ejemplos de código que se muestran en el documento.

ToolBox: clase con varios métodos que pueden ser de ayuda (tratamiento de flujo de datos, almacén de claves, etc.).

Paquete *es.gob.fnmt.gui.fragment*

NFCCommunicationFragment: clase que hereda de *es.gob.fnmt.nfc.NFCCommReaderFragment* y por tanto adquiere toda las características funcionales de ésta, añadiendo además una interfaz gráfica que ayuda al usuario en la interacción del DNIe con su dispositivo móvil. Así, los requisitos en la inicialización son los mismos que la clase padre, con la opción de configurar la apariencia según las necesidades.

La interfaz gráfica de este *Fragment* presenta una animación que responde a los eventos de comunicación del DNIe con el dispositivo, una barra de progreso y dos cajas de texto que se pueden editar para mostrar la información que interese. La función *updateInfo()* permite esta edición, y deberá ser invocada para actualizar también el estado de la animación. Por defecto, si no se indican textos se mostrarán mensajes predefinidos según el evento. Un ejemplo de uso se muestra en el siguiente código, disponible en *SampleActivity.connect_gui*.

```
1  @Override
2  public void doNotify(final NFC_callback_notify notify, String msg) {
3      String message =msg;
4      switch (notify){
5          case NFC_TASK_INIT:
6              _readerFragment.updateInfo(notify,"Comunicando con Dnie",
7                  ↪ "estableciendo canal seguro...");
8              runOnUiThread(new Runnable() {
9                  @Override
10                 public void run() {
11                     resultInfo.setVisibility(GONE);
12                 }
13             });
14 }
```

4 INTEGRACIÓN EN APLICACIONES ANDROID DNIEDROID v2.2

```
13         break;
14     case NFC_TASK_UPDATE:
15         _readerFragment.updateInfo(notify, "Comunicando con Dnie",
16         ↪ "obteniendo información...");
17         break;
18     case NETWORKCOMM_TASK_INIT:
19         message = "Connectando con sitio web...";
20         break;
21     case NETWORKCOMM_TASK_DONE:
22         message = "Conexión finalizada.";
23         break;
24     case ERROR:
25         if(msg.equalsIgnoreCase(NFCCommReaderFragment.ERROR_PIN_LOCKED)) {
26             Toast.makeText(this.getApplicationContext(),
27             ↪ getString(R.string.lib_dni_password_error_pin_locked),
28             ↪ Toast.LENGTH_LONG).show();
29
30             FragmentTransaction transaction =
31             ↪ getFragmentManager().beginTransaction();
32             transaction.remove(_readerFragment);
33             transaction.commit();
34             return;
35         }
36         else {
37             _readerFragment.updateInfo(notify, "Error en comunicación",
38             ↪ message);
39             if (msg.contains("CAN incorrecto")) {
40                 Toast.makeText(this.getApplicationContext(), msg,
41                 ↪ Toast.LENGTH_LONG).show();
42
43                 FragmentTransaction transaction =
44                 ↪ getFragmentManager().beginTransaction();
45                 transaction.remove(_readerFragment);
46                 transaction.commit();
47                 return;
48             }
49         }
50         break;
51     default:
52         _readerFragment.updateInfo(notify, "Comunicando con Dnie",
53         ↪ message);
54 }
55
56 if(notify == NFC_callback_notify.NFC_TASK_FINISHED){
57     ProgressDialog myProgressDialog = new ProgressDialog(this);
```

DNIEDROID v2.2 4 INTEGRACIÓN EN APLICACIONES ANDROID

```
50
51     myProgressDialog.getWindow().setBackgroundDrawable(new
        ↳ ColorDrawable(Color.TRANSPARENT));
52     myProgressDialog.setTitle("Descargando datos");
53     myProgressDialog.setMessage("");
54     myProgressDialog.setProgress(0);
55     myProgressDialog.setMax(100);
56
57     _myNetProgressDialog = new ProgressDialogUI.Builder(myProgressDialog)
58         .contentLayout(R.layout.progress)
59         .progressBar(R.id.externalProgressRead)
60         .title(R.id.title)
61         .description(R.id.messages)
62         .build();
63
64     _networkFragment = new NetworkCommunicationFragment();
65     _networkFragment.setDialog(_myNetProgressDialog, false);
66
67     FragmentTransaction transaction =
        ↳ getFragmentManager().beginTransaction();
68     transaction.replace(R.id.fragment_container, _networkFragment);
69     transaction.addToBackStack(null);
70     transaction.commit();
71 }
72 }
```

NetworkCommunicationFragment: clase que hereda de *android.app.Fragment* y que simplemente se ofrece como una mera interfaz gráfica mientras se realizan los procesos de comunicación a través de la red. Al igual que la clase del mismo paquete, visualmente proporciona una imagen, no animada en este caso, una barra de progreso y dos cajas de texto que muestran mensajes predefinidos. Estos elementos, al contrario que en la clase anterior, no se pueden editar. Para el uso de la clase se incluye una interfaz *NetworkCommunicationFragment.NetCommFragmentListener* con dos funciones: *netConnDownload()* y *netConnDone()*.

La primera se invoca como inicio del proceso principal de descarga de datos, utilizando por ejemplo la clase *es.gob.fnmt.net.ssl.DNIESSLSocketFactory* descrita anteriormente. Se muestra un ejemplo de implementación a continuación, disponible en *SampleActivity_connect_gui*.

```
1     @Override
2     public void netConnDownload() throws IOException {
```

4 INTEGRACIÓN EN APLICACIONES ANDROID DNIEDROID v2.2

```
3      HTMLParser parser = new HTMLParser(getApplicationContext()); //clase
    ↪ de ayuda para obtener información de las páginas HTML.
4      Map<String,String> postParams = new HashMap<>(); //Lista de parámetros
    ↪ a pasar a la petición 'POST'.
5      try {
6          SSLSocketFactory dniesSLSocketFactory = DNIESSLSocketFactory.getInstanc
    ↪ e(NFCCCommReaderFragment.getKeyStore(), //almacén del DNIE
    ↪ con los certificados de usuario.
7          Toolbox.getKeyStoreFromResource(R.raw.truststore,
    ↪ getApplicationContext(), //almacén con los certificados de CA
    ↪ reconocidos para la conexión con los servidores.
8          new KeyManagerPolicy.Builder().init().addAlias(DnieKeyStore.AUTH_CERT_
    ↪ ERT_ALIAS).build(), //política de selección de certificados
    ↪ de usuario. En este caso indicamos el certificado de
    ↪ autenticación.
9          getApplicationContext());
10
11      llamada = 0;
12      updateProgressDialog("Descargando datos...", "Llamada " + ++llamada);
13
14      parser.setContent(new HTTPClient(URL_SS_LOGIN).getStringResponse());
15      Element dniesElement = parser.getDocument().getElementById("loginCertific
    ↪ adoFormSubmit").parent().getElementsByTag("input").first();
16      String finalUrl = null;
17
18      postParams.put("loginCertificadoFormSubmit",
    ↪ dniesElement.getElementsByAttributeValue("name",
    ↪ "loginCertificadoFormSubmit").first().val());
19      finalUrl = URL_TUSS_DOMAIN + dniesElement.parent().attr("action");
20      updateProgressDialog(null, "Llamada " + ++llamada);
21
22      parser.setContent(new HTTPClient(finalUrl,
    ↪ postParams).getStringResponse(dniesSSLSocketFactory));
23      updateProgressDialog(null, "Llamada " + ++llamada);
24
25      Element formMovil =
    ↪ parser.getDocument().getElementById("formCapturaMovil");
26      if (formMovil != null) {
27          finalUrl = URL_TUSS_DOMAIN + formMovil.attr("action").trim() +
    ↪ "?prefijo=&movil=&tipoAccionCapDatosSol=CANCELAR";
28          parser.setContent(new
    ↪ HTTPClient(finalUrl).getStringResponse(dniesSSLSocketFactory));
29      }
30
```

DNIEDROID v2.2 4 INTEGRACIÓN EN APLICACIONES ANDROID

```
31 //Parece que hay un timeout en el cálculo de cotización, del que
   ↳ dependen varias opciones (histórico, ect.)
32 if(parser.getDocument().getElementById("urlRecarga")!=null) {
33     String recarga =
   ↳ parser.getDocument().getElementById("urlRecarga").val();
34     String paginaActual =
   ↳ parser.getDocument().getElementById("paginaActual").val();
35     int paso = 0;
36     String htmlContent;
37     do {
38         updateProgressDialog(null,"Llamada "+ ++llamada);
39         paso++;
40         htmlContent = new HTTPClient(URL_TUSS_DOMAIN + recarga +
   ↳ "?paso=" + paso).getStringResponse(dnieSSLSocketFactory);
41     }
42     while(paso<3&&!htmlContent.contains("calculado"));
43     htmlContent = new HTTPClient(URL_TUSS_DOMAIN + paginaActual +
   ↳ "?paso=3").getStringResponse(dnieSSLSocketFactory);
44     //Completamos el contenido de la página ppal con los datos
   ↳ calculados
45     parser.getDocument().getElementById("trabPen_calculando_to_home").]
   ↳ html(htmlContent);
46 }
47
48 parser.removeLinks();
49 Document document = (Document)parser.getDocument();
50
51 _htmlBody = HTML_SAMPLE.replace("##COTIZADO##",document.getElementsB]
   ↳ yClass("infoButBox__item").get(0).outerHtml()).
52 replace("##JUBILACION##",document.getElementsByClass("infoButBox__it]
   ↳ em").get(1).outerHtml());
53 }
54 catch(Exception e){
55     throw new IOException(e);
56 }
57 }
```

La función *netConnDone()* se invoca en el ejemplo cuando termina el proceso de descarga de datos, permitiendo así tratar el siguiente paso que será mostrar la información o tratar el error si lo hubiera. Un ejemplo de implementación de la función se muestra en el siguiente código.

4 INTEGRACIÓN EN APLICACIONES ANDROID

DNIEDROID v2.2

```
1      @Override
2      public void netConnDone(boolean error) {
3          AlertDialog.Builder alert = new
4              ↳ AlertDialog.Builder(SampleActivity_2.this);
5          alert.setNegativeButton("Salir", new DialogInterface.OnClickListener()
6              ↳ {
7              @Override
8              public void onClick(DialogInterface dialog, int id) {
9                  setCan.setVisibility(VISIBLE);
10                 dialog.dismiss();
11             }
12         });
13     try {
14         if(!error) {
15             alert.setTitle("Tu seguridad social");
16
17             WebView webView = new WebView(SampleActivity_2.this);
18             webView.setInitialScale(1);
19             webView.getSettings().setSupportZoom(true);
20             webView.getSettings().setBuiltInZoomControls(true);
21             webView.getSettings().setJavaScriptEnabled(true);
22             webView.getSettings().setUseWideViewPort(true);
23
24             webView.getSettings().setAllowContentAccess(true);
25             webView.getSettings().setAllowFileAccess(true);
26             webView.getSettings().setAllowFileAccessFromFileURLs(true);
27             webView.getSettings().setAllowUniversalAccessFromFileURLs(true);
28             webView.loadData(_SSLresultado, "text/html;
29                 ↳ charset=utf-8","utf-8");
30             alert.setView(webView);
31         }
32         else{
33             alert.setTitle("Error en comunicaciones");
34             alert.setTitle("Se ha producido un error en la petición de
35                 ↳ información.");
36         }
37     }catch (Exception e){
38         alert.setTitle("Error en aplicación:");
39         alert.setTitle("Se ha generado una excepción:"+e.getMessage());
40     }
41     alert.show();
42 }
```

Paquete *es.gob.fnmt.gui*

ProgressDialogUI: clase que permite establecer la apariencia de los elementos visuales de los *Fragments* del paquete anterior, así como incorporar un cuadro de diálogo de progreso que sustituya al de por defecto. De esta manera, se puede adaptar la apariencia y gestionar los mensajes que se muestran como el desarrollador considere. Para su construcción, la propia clase provee un *Builder*. Para establecer el nuevo cuadro de diálogo se utilizará la función *setDialog()* en el *Fragment* correspondiente, indicando además si se quieren mostrar en éste los mensajes predefinidos.

Definiendo los objetos y estableciendo los parámetros en la instancia del *Fragment*. Ejemplo de código en *SampleActivity_connect_gui*:

```
1   ProgressDialogUI _myNetProgressDialog = null; //Miembro de clase
2   ...
3   if(notify == NFC_callback_notify.NFC_TASK_FINISHED){
4       ProgressDialog myProgressDialog = new ProgressDialog(this);
5
6       myProgressDialog.getWindow().setBackgroundDrawable(new
           ↳ ColorDrawable(Color.TRANSPARENT));
7       myProgressDialog.setTitle("Descargando datos");
8       myProgressDialog.setMessage("");
9       myProgressDialog.setProgress(0);
10      myProgressDialog.setMax(100);
11
12      _myNetProgressDialog = new ProgressDialogUI.Builder(myProgressDialog)
13          .contentLayout(R.layout.progress)
14          .progressBar(R.id.externalProgressRead)
15          .title(R.id.title)
16          .description(R.id.messages)
17          .build();
18
19      _networkFragment = new NetworkCommunicationFragment();
20      _networkFragment.setDialog(_myNetProgressDialog,false);
21
22      FragmentTransaction transaction =
           ↳ getFragmentManager().beginTransaction();
23      transaction.replace(R.id.fragment_container, _networkFragment);
24      transaction.addToBackStack(null);
25      transaction.commit();
26  }
```

4 INTEGRACIÓN EN APLICACIONES ANDROID DNIEDROID v2.2

Por ejemplo, en el código anterior se podría realizar esta modificación para mostrar un cuadro de progreso propio.

```
1    ...
2    private void updateProgressDlg(final String title, final String msg){
3        runOnUiThread(new Runnable() {
4            @Override
5            public void run() {
6                if(title!=null){
7                    _myNetProgressDialog.setTitle(title);
8                }
9                if(msg!=null) {
10                   _myNetProgressDialog.setMessage(msg);
11                   _myNetProgressDialog.incrementProgressBy(17);
12                }
13            }
14        });
15    }
16    public void netConnDownload() throws IOException {
17        try {
18            SSLSocketFactory dniesSLSocketFactory = DNIESSLSocketFactory.getInst
19            ↪ ance(NFCCCommReaderFragment.getKeyStore(), //almacén del DNIE
20            ↪ con los certificados de usuario.
21            Toolbox.getKeyStoreFromResource(R.raw.truststore,
22            ↪ getApplicationContext(), //almacén con los certificados de CA
23            ↪ reconocidos para la conexión con los servidores.
24            new KeyManagerPolicy.Builder().init().addAlias(DnieKeyStore.AUTH_C
25            ↪ ERT_ALIAS).build(), //política de selección de certificados
26            ↪ de usuario. En este caso indicamos el certificado de
27            ↪ autenticación.
28            getApplicationContext());
29
30            llamada = 0;
31            updateProgressDlg("Descargando datos...", "Llamada "+ ++llamada);
32
33            parser.setContent(new HTTPClient(URL_SS_LOGIN).getStringResponse());
34            Element dniesElement = parser.getDocument().getElementById("loginCert
35            ↪ ificadoFormSubmit").parent().getElementsByTag("input").first();
36            String finalUrl = null;
37
38            postParams.put("loginCertificadoFormSubmit",
39            ↪ dniesElement.getElementsByAttributeValue("name",
40            ↪ "loginCertificadoFormSubmit").first().val());
```

DNIEDROID v2.2 4 INTEGRACIÓN EN APLICACIONES ANDROID

```
31     finalUrl = URL_TUSS_DOMAIN + dniesElement.parent().attr("action");
32     updateProgressDlg(null, "Llamada "+ ++llamada);
33
34     parser.setContent(new HTTPClient(finalUrl,
35     ↪ postParams).getStringResponse(dniesSSLSocketFactory));
36     updateProgressDlg(null, "Llamada "+ ++llamada);
37
38     Element formMovil =
39     ↪ parser.getDocument().getElementById("formCapturaMovil");
40     if (formMovil != null) {
41         finalUrl = URL_TUSS_DOMAIN + formMovil.attr("action").trim() +
42         ↪ "?prefijo=&movil=&tipoAccionCapDatosSol=CANCELAR";
43         parser.setContent(new
44         ↪ HTTPClient(finalUrl).getStringResponse(dniesSSLSocketFactory));
45     }
46
47     //Parece que hay un timeout en el cálculo de cotización, del que
48     ↪ dependen varias opciones (histórico, ect.)
49     if(parser.getDocument().getElementById("urlRecarga")!=null) {
50         String recarga =
51         ↪ parser.getDocument().getElementById("urlRecarga").val();
52         String paginaActual =
53         ↪ parser.getDocument().getElementById("paginaActual").val();
54         int paso = 0;
55         String htmlContent;
56         do {
57             updateProgressDlg(null, "Llamada "+ ++llamada);
58             paso++;
59             htmlContent = new HTTPClient(URL_TUSS_DOMAIN + recarga +
60             ↪ "?paso=" + paso).getStringResponse(dniesSSLSocketFactory);
61         }
62         while(paso<3&&!htmlContent.contains("calculado"));
63         htmlContent = new HTTPClient(URL_TUSS_DOMAIN + paginaActual +
64         ↪ "?paso=3").getStringResponse(dniesSSLSocketFactory);
65         //Completamos el contenido de la página ppal con los datos
66         ↪ calculados
67         parser.getDocument().getElementById("trabPen_calculando_to_home").
68         ↪ html(htmlContent);
69     }
70
71     parser.removeLinks();
72     Document document = (Document)parser.getDocument();
73
74     _htmlBody = HTML_SAMPLE.replace("##COTIZADO##",document.getElementsB
75     ↪ yClass("infoButBox__item").get(0).outerHtml()).
```

4 INTEGRACIÓN EN APLICACIONES ANDROID DNIEDROID v2.2

```
64         replace("##JUBILACION##", document.getElementsByClassName("infoButBox__it_
        ↪ em").get(1).outerHtml());
65     }
66     catch(Exception e){
67         throw new IOException(e);
68     }
69 }
70 ...
```

PasswordUI: clase que establece la apariencia de los elementos visuales del cuadro de diálogo de petición de PIN. Se puede incorporar un cuadro de diálogo de petición de PIN propio que sustituya al de por defecto (ver más adelante en *es.gob.jmulticard.ui.passwordcallback*).

```
1 PasswordUI.setTextColor(Color.YELLOW);
2 PasswordUI.setBackgroundColor(Color.BLUE);
3 PasswordUI.setTextTypeFace(Typeface.createFromAsset(...));
```

CertificateUI: clase que establece la apariencia de los elementos visuales de la ventana de diálogo de selección de certificado. Por defecto, esta ventana no aparece cuando las políticas de selección de certificados establecidas, es decir *es.gob.fnmt.policy.KeyManagerPolicy*, sólo filtran un certificado.

Paquete *es.gob.fnmt.policy*

KeyManagerPolicy: clase que permite indicar las políticas de selección de certificado cuando se va a realizar una operación de firma. Se puede añadir políticas por alias conocido, uso de clave o uso extendido de clave (definidas en *KeyManagerPolicy.ExtendedKeyUsage* y *KeyManagerPolicy.KeyUsage*). Para su construcción la propia clase provee un *Builder*.

```
1 KeyManagerPolicy keyManagerPolicy = new KeyManagerPolicy.Builder().init(
    ↪ ).addAlias(DnieKeyStore.AUTH_CERT_ALIAS)
    ↪ .addKeyUsage(KeyManagerPolicy.KeyUsage.digitalSignature).build();
```

DNIEDROID v2.2 4 INTEGRACIÓN EN APLICACIONES ANDROID

Paquete *es.gob.jmulticard.ui.passwordcallback*

DialogUIHandler: interfaz a implementar para proporcionar un cuadro de diálogo de petición de PIN a la clase *PasswordUI* en el método *setPasswordDialog()*, si se quiere sustituir el de por defecto. En el código siguiente se muestra la línea de ejemplo.

```
1 PasswordUI.setPasswordDialog(new MyPasswordDialog());  
    ↪ //'MyPasswordDialog' implementa la interfaz 'DialogUIHandler'
```

El ejemplo completo se encuentra en *SampleActivity_connect_gui*.

Paquete *de.tsenger.androsmex.mrtd*

DG1_Dnie, DG2, DG7, DG11, DG13, EF_COM: clases que encapsulan la información devuelta por las funciones correspondientes de la clase *es.gob.jmulticard.jse.provider.DnieKeyStore*. Se muestra un ejemplo de acceso a los datos en el código siguiente, disponible en *SampleActivity_read_data*.

```
1 DG1_Dnie data1 = ((DnieKeyStore)keyStore).getDatagroup1();  
2 DG2 data2 = ((DnieKeyStore)keyStore).getDatagroup2();  
3 DG7 data7 = ((DnieKeyStore)keyStore).getDatagroup7();  
4 DG11 data11 = ((DnieKeyStore)keyStore).getDatagroup11();  
5 DG13 data13 = ((DnieKeyStore)keyStore).getDatagroup13();  
6 EF_COM dataef = ((DnieKeyStore)keyStore).getEF_COM();  
7  
8 data1.getDateOfBirth(); //Obtenemos la fecha de nacimiento como cadena  
    ↪ en formato 'yyMMdd'  
9 data2.getImageBytes(); //Obtenemos la imagen de la cara  
10 data7.getImageBytes(); //Obtenemos la imagen de la firma manuscrita
```

4.5. Comprobación de edad

La librería permite realizar una consulta directa para verificar la edad del propietario del DNIe. Para ello se envía la fecha de interés para comprobar si la fecha de nacimiento del sujeto no es posterior a ésta. Esto permite verificar diferentes edades según sea la necesidad. Así, la función devolverá un valor

4 INTEGRACIÓN EN APLICACIONES ANDROID DNIEDROID v2.2

booleano 'verdadero' si el nacimiento del sujeto es anterior o en la fecha indicada, o 'falso' si es posterior. Por ejemplo, si se quiere comprobar si el sujeto es mayor (o igual) de 18 años a día 12 de mayo del 2018 (12/05/2018), se enviará al DNIe la fecha 12 de mayo del 2000 (12/05/2000). Si queremos saber si es mayor de 16, se enviará la fecha 12 de mayo del 2002.

Para su invocación es necesario haber instanciado previamente un objeto *es.gob.jmulticard.jse.provider.DnieKeyStore* como se ha visto en ejemplos anteriores. En el Activity *SampleActivity_age_verification* de la aplicación de ejemplo se puede observar el uso de esta función.

```
1  ...
2  KeyStore keyStore = new DnieKeyStore(new
    ↳ MrtdKeyStoreImpl(_can.getCanNumber(), tag), p);
3  keyStore.load(null, null);
4
5  updateInfo("Leyendo datos...", "Verificando edad...");
6  final Calendar calendar = Calendar.getInstance();
7  calendar.set(_date.getYear(), _date.getMonth(), _date.getDayOfMonth());
8  _image.setCensored(((DnieKeyStore)keyStore).verifyAge(calendar.getTime()
    ↳ )? CanvasView.CENSORED.FALSE:
    ↳ CanvasView.CENSORED.TRUE);
9  // Actualizamos la imagen a mostrar
10 updateInfo("Comprobar edad", null);
11 ...
```

4.6. Obtener estado del certificado

Para obtener el estado de revocación de los certificados digitales del DNIe es necesario utilizar el servicio OSCP [8] indicado en la extensión *Acceso a información de autoridad* o **AIA** (Authority Information Access), dentro del propio certificado. En el proyecto con el código fuente de ejemplos se implementa la clase *com.fnmt.sample-dnie-app.utils.pki.OSCP* que encapsula la funcionalidad de obtención de estado de revocación del certificado. Un ejemplo de uso se puede encontrar en *SampleActivity-provider*.

```
1  ...
2  CertificateStatus status =
    ↳ OSCP.checkCertificateStatus(_certificateChain[0],
    ↳ _certificateChain[1], "http://ocsp.dnie.es");
```

```
3     if(status == CertificateStatus.GOOD || status instanceof UnknownStatus){  
4         updateInfo("Realizando firma...", null);  
5         _signature = getSignature(_privateKey);  
6     }  
7     ...
```

Este es sólo un código de ejemplo. Las políticas de actuación según la información recuperada del servicio OSCP corresponden al desarrollador (en este caso aunque se desconozca el estado del certificado, se realiza la firma).

4.7. Información adicional

Para más información acerca de la funcionalidad disponible, las clases y sus usos, se dispone de documentación en formato *JavaDoc*, así como de un proyecto para *Android Studio* donde se puede encontrar el código utilizado para los ejemplos del documento.

Referencias

- [1] D. COOPER, S. SANTESSON, S. FARRELL, S. BOEYEN, R. HOUSLEY y W. POLK, *RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, 2008.
- [2] ISO/IEC JTC 1/SC 17, *ISO/IEC 14443. Identification cards – Contactless integrated circuit cards – Proximity cards.*, 3ª edición, 2016.
- [3] NFC FORUM, NXP SEMICONDUCTORS, *NFC Forum Type Tags White Paper V1.0*, 2009.
- [4] *CWA 14890-1: Application Interface for smart cards used as Secure Signature Creation Device*, 2009.
- [5] ICAO, *DOC 9303-11, Machine Readable Travel Documents Part 11-Security Mechanisms for MRTDs*, 2015.
- [6] ORACLE, *Java Cryptography Architecture (JCA) Reference Guide*.
- [7] DPTO. DOCUMENTOS DE IDENTIFICACIÓN Y TARJETAS, FNMT, *Guía de referencia del DNIe con NFC*, 2017.
- [8] S. SANTESSON, M. MYERS, R. ANKNEY, A. MALPANI, S. GALPERIN y C. ADAMS, *RFC 6960, X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*, 2013.