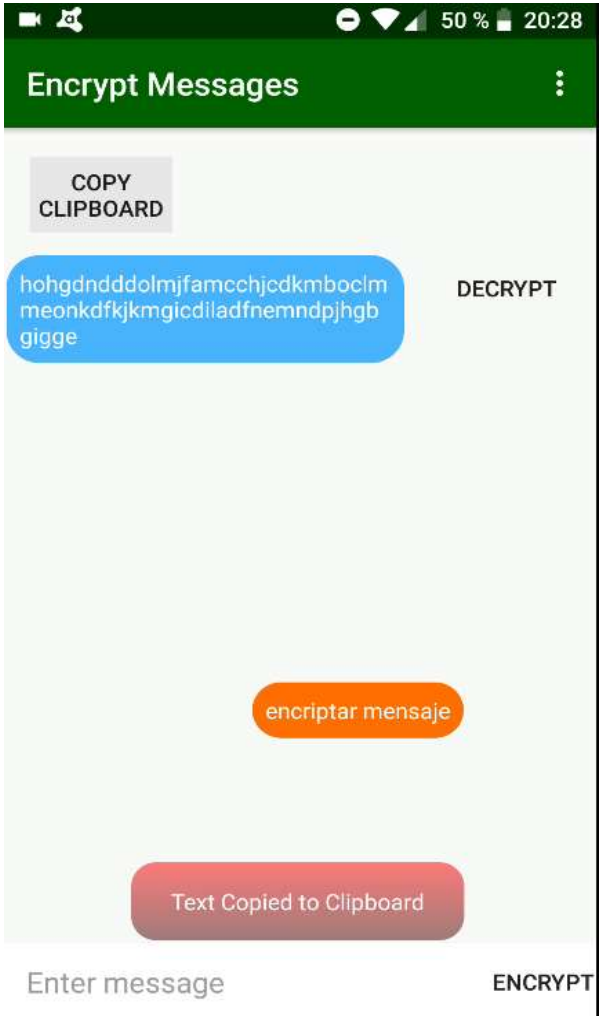
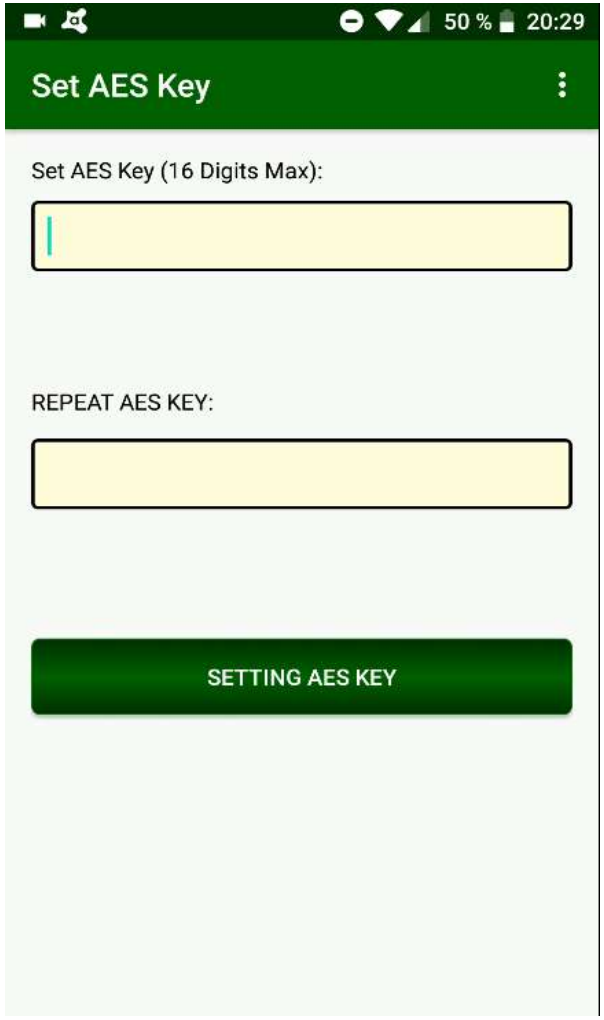
DESARROLLO DE CÓDIGO DE SEGURIDAD EN ANDROID

CONCURRENCIA

- **Formas de Implementar la concurrencia:**
  - **Procesos**
    - **Sistemas Operativos unix (fork), windows**
    - **Lenguaje c**
  - **Hilos**
    - **Sistemas Operativos unix, windows, Android**
    - **Lenguajes c, java**
  - **Llamadas asíncronas desde un único proceso (Select)**
    - **Sistemas Operativos unix, windows**
    - **Lenguaje c, c# (.Net Framework)**
  - **Asynctask, IntentService (Android)**

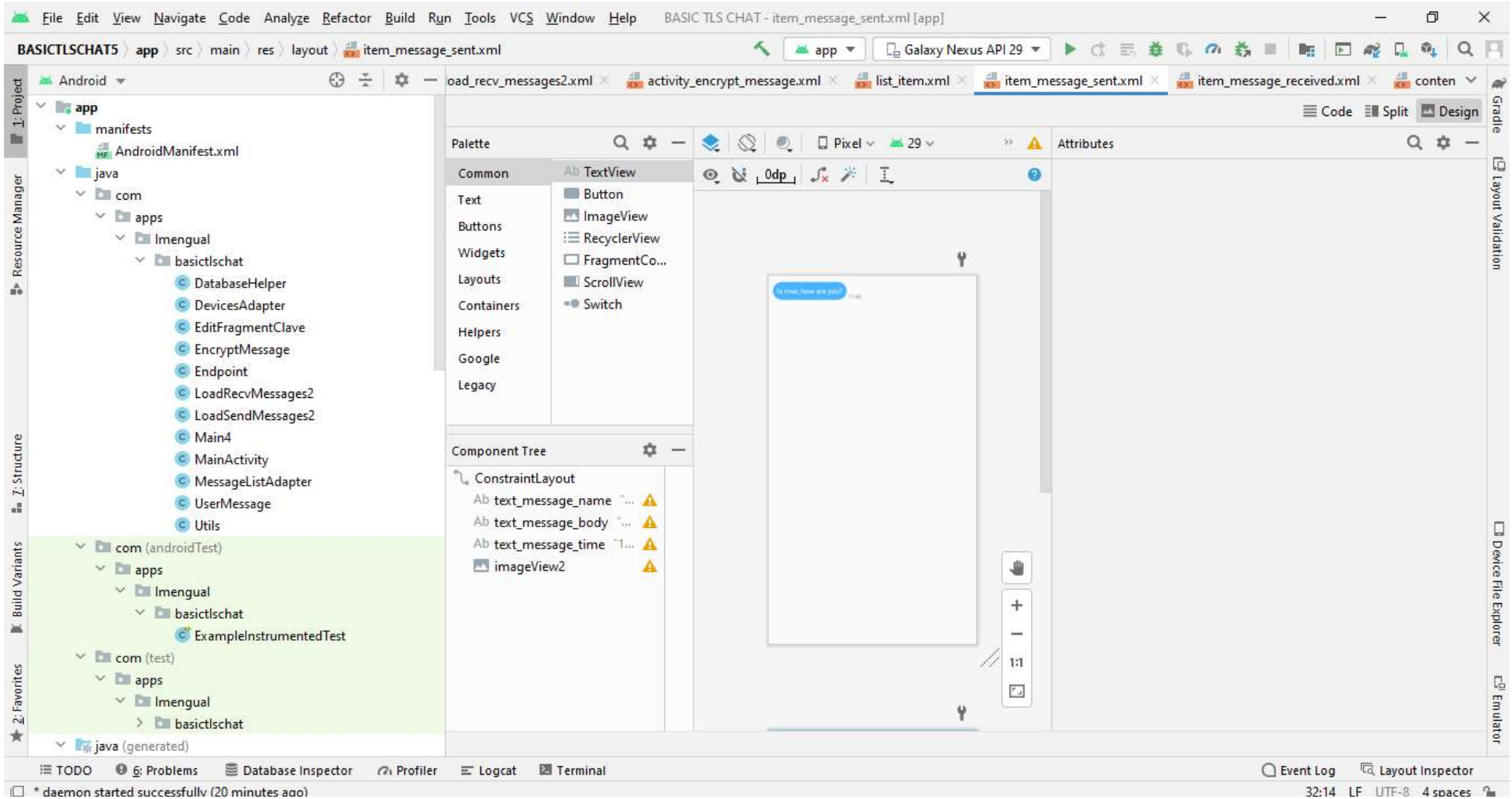
APP BASIC TLS CHAT





APP BASIC TLS CHAT

Código Fuente



CLIENTE SERVIDOR SIN SSL

```
mythread=new Thread(new Runnable() {
        public void run() {
        try {
            ServerSocket socServer = new ServerSocket(5223);
           System.out.println ("SERVIDOR ARRANCADO");
          Socket socClient = null;

           while (true) {
                    socClient = socServer.accept();
                     int puerto_remoto=socClient.getPort();
                     System.out.println("puert remoto ....... "+puerto_remoto );
                     InetAddress peer_adress= socClient.getInetAddress();
                              ...................................
              //For each client new instance of AsyncTask will be created
              ServerAsyncTask serverAsyncTask = new ServerAsyncTask();
              //Start the AsyncTask execution
              //Accepted client socket object will pass as the parameter
              serverAsyncTask.execute(new Socket[]{socClient});
           }
        } catch (IOException e) {            e.printStackTrace();          }
      }
    });
    mythread.start();
```

```
class ServerAsyncTask extends AsyncTask<Socket, Void, String[]> {
    //Background task which serve for the client
    @Override
    protected String[] doInBackground(Socket... params) {
        String[] result = new String[10];
      //Get the accepted socket object
      Socket mySocket = params[0];
      String string = null;

      try {

          OutputStream Flujo_salida = mySocket.getOutputStream();
          InputStream Flujo_entrada = mySocket.getInputStream();
          DataOutputStream Flujo_s = new DataOutputStream(Flujo_salida);
          DataInputStream Flujo_e = new DataInputStream(Flujo_entrada);

          mensaje_rev2 = Flujo_e.readUTF();
           result[0]=mensaje_rev2;
          mySocket.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
      return result;
    }
```

Para retornar resultados se usa el método:
void onPostExecute(String[] s) {…}

```
class ClientAsyncTask extends AsyncTask<String, Void, String> {
   @Override    protected String doInBackground(String... params) {
    String result = null;
    try {
            //Get the input stream of the client socket
             System.out.println("antes de newsocket");
            Socket socket = new Socket(myip2,5223);
            OutputStream Flujo_salida = socket.getOutputStream();
            InputStream Flujo_entrada = socket.getInputStream();
            DataOutputStream Flujo_s = new DataOutputStream(Flujo_salida);
            DataInputStream Flujo_e = new DataInputStream(Flujo_entrada);
             EditText EditTextEnvio= (EditText)findViewById(R.id.editText2);
           String TextoEnvio= String.valueOf(EditTextEnvio.getText());
             Flujo_s.writeUTF(TextoEnvio);
          //Close the client socket
          socket.close();
    } catch (NumberFormatException e) {
     e.printStackTrace();
    } catch (UnknownHostException e) {      e.printStackTrace();      }
     catch (IOException e) {      e.printStackTrace();      }
    return result;
   }
```

Para retornar resultados se usa el método:
void onPostExecute(String[] s) {…}

CLIENTE SERVIDOR CON SSL

CARGA MANUAL ALMACENES

Almacenes de Seguridad en *Android*

- **Los Almacenes de Seguridad en *Android* deben ser cargados de forma manual. No puede usarse *System.setProperty(…).***
- **Los Almacenes de Seguridad en *Android* no pueden ser del tipo *JKS (Java Keystore).* Tienen que ser del tipo *BKS (Bouncy Castle Provider).***
- **Estos almacenes se tienen que crear con código fuente *java* añadiendo la *librería BKS* (no se puede utilizar la herramienta *Keytool)*.**

Carga de Almacenes en *Android*

- **Los almacenes se cogen del path**
  - **C:\Users\lmengual\AndroidStudioProjects\MyApplication22\app\src\main\res\raw**
- **Con la instrucción:**
  - **InputStream is = getResources().openRawResource(R.raw.almacensr);**

```
mythread=new Thread(new Runnable() {
  public void run() {
   try {
        SSLContext ctx;
        KeyManagerFactory kmf, kmf2;
        KeyStore ks, ks2;
         System.setProperty("javax.net.debug","ssl");
        char[] fraseclave = "oooooo".toCharArray();
       //  Security.addProvider(new BouncyCastleProvider());  // Cargar el provider BC
        ks = KeyStore.getInstance("BKS");
        kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
        InputStream is = getResources().openRawResource(R.raw.almacensr);
         ks.load(is, fraseclave);
        kmf.init(ks, fraseclave);
        ctx = SSLContext.getInstance("TLS");
         ctx.init(kmf.getKeyManagers(), null, null);

        SSLServerSocketFactory sslserversocketfactory = ctx.getServerSocketFactory();
        SSLServerSocket sslserversocket =(SSLServerSocket) sslserversocketfactory.createServerSocket(5223);
        System.out.println("SERVIDOR ECO SSL ESPERANDO PTO 9999 ..... ");
```

```
            //Infinite loop will listen for client requests to connect
            while (true) {
                //Accept the client connection and hand over communication to server side client socket
                sslsocket = (SSLSocket) sslserversocket.accept();
               SSLSession sesion = sslsocket.getSession();
               //For each client new instance of AsyncTask will be created
                ServerAsyncTask serverAsyncTask = new ServerAsyncTask();
                //Start the AsyncTask execution
                //Accepted client socket object will pass as the parameter
                serverAsyncTask.execute(new Socket[]{sslsocket});
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (CertificateException e) {
            e.printStackTrace();
        } catch (UnrecoverableKeyException e) {
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (KeyStoreException e) {
            e.printStackTrace();
        } catch (KeyManagementException e) {            e.printStackTrace();            }
        }
    });
    mythread.start();
// .start();
```

```
class ServerAsyncTask extends AsyncTask<Socket, Void, String[]> {
    //Background task which serve for the client
    @Override      protected String[] doInBackground(Socket... params) {

        System.out.println("entro tarea asincrona:");
        String[] result = new String[2];
        SSLSocket mySocket = (SSLSocket) params[0];
        String string = null;
        try {

            OutputStream Flujo_salida = mySocket.getOutputStream();
            InputStream Flujo_entrada = mySocket.getInputStream();
            DataOutputStream Flujo_s = new DataOutputStream(Flujo_salida);
            DataInputStream Flujo_e = new DataInputStream(Flujo_entrada);

             mensaje_rev2 = Flujo_e.readUTF();
             System.out.println("Mensaje Recibido:"+mensaje_rev2);
            mySocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return result;
    }
```

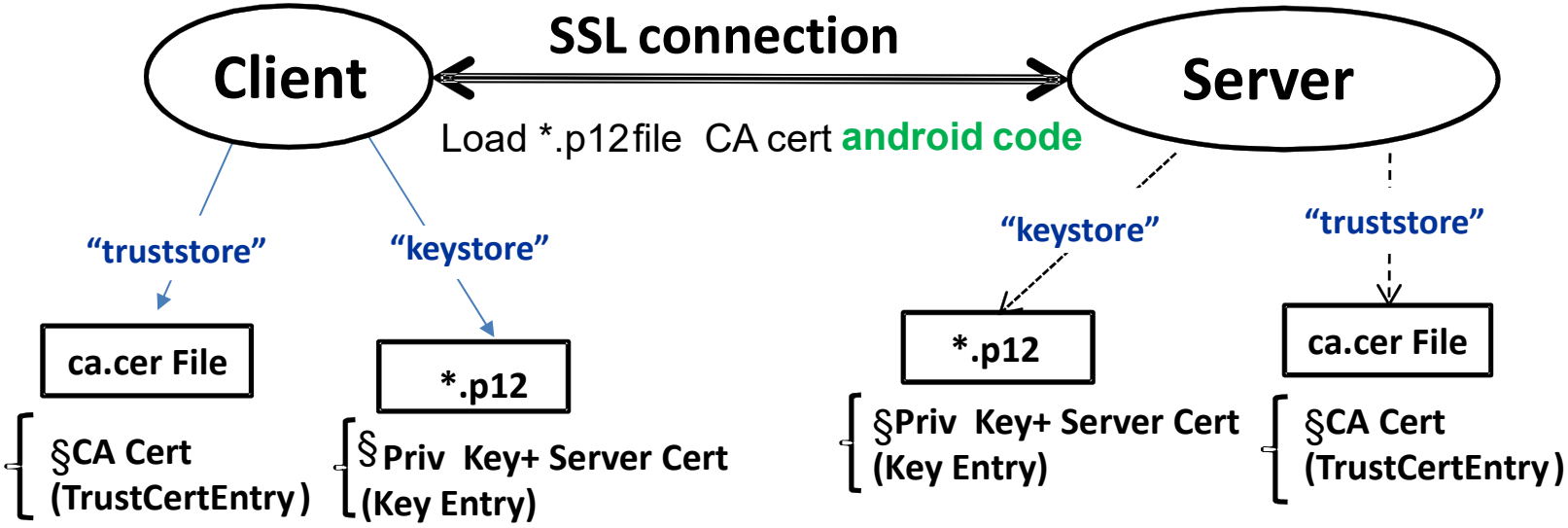
```
class ClientAsyncTask extends AsyncTask<String, Void, String> {
    @Override     protected String doInBackground(String... params) {
     String result = null;
     try {

       EditText dirip= (EditText)findViewById(R.id.editText);
       String myip2= String.valueOf(dirip.getText());
        SSLContext ctx;
        KeyManagerFactory kmf, kmf2;
        KeyStore ks, ks2;

        char[] fraseclave = "oooooo".toCharArray();
        kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
        ks = KeyStore.getInstance("BKS");
        ctx = SSLContext.getInstance("TLS");
        InputStream is3 = getResources().openRawResource(R.raw.almacentrustcl);
        ks.load(is3, fraseclave);
        kmf.init(ks, fraseclave);
        TrustManagerFactory tmf =
           TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        tmf.init(ks);
```

```
    ctx.init(null, tmf.getTrustManagers(), null);
    SSLSocketFactory sslsocketfactory = ctx.getSocketFactory();
    SSLSocket sslsocket = (SSLSocket) sslsocketfactory.createSocket(myip2, 5223);
    SSLSession sesion = sslsocket.getSession();

    OutputStream Flujo_salida = sslsocket.getOutputStream();
    InputStream Flujo_entrada = sslsocket.getInputStream();
    DataOutputStream Flujo_s = new DataOutputStream(Flujo_salida);
    DataInputStream Flujo_e = new DataInputStream(Flujo_entrada);

   EditText EditTextEnvio= (EditText)findViewById(R.id.editText2);
    String TextoEnvio= String.valueOf(EditTextEnvio.getText());
    Flujo_s.writeUTF(TextoEnvio);

//Close the client socket
    sslsocket.close();
} catch (NumberFormatException e) {
 e.printStackTrace();
} catch (UnknownHostException e) {
 e.printStackTrace();
} catch (IOException e) {
 e.printStackTrace();
} catch (CertificateException e) {   ................
```

Actualización de un *Socket* convencional a *Socket SSL*

```
if (ssl) {

System.out.println("SR:Entro por if ssl");
SSLSocketFactory sslSf = ctx.getSocketFactory();

//Actualizamos el socketm para que ahora sea ssl
sslsocket = (SSLSocket) sslSf.createSocket(socket, null, socket.getPort(), false);

……

}
```

Aplicaciones TLS en Android con ficheros *.p12 y *.cer (I)

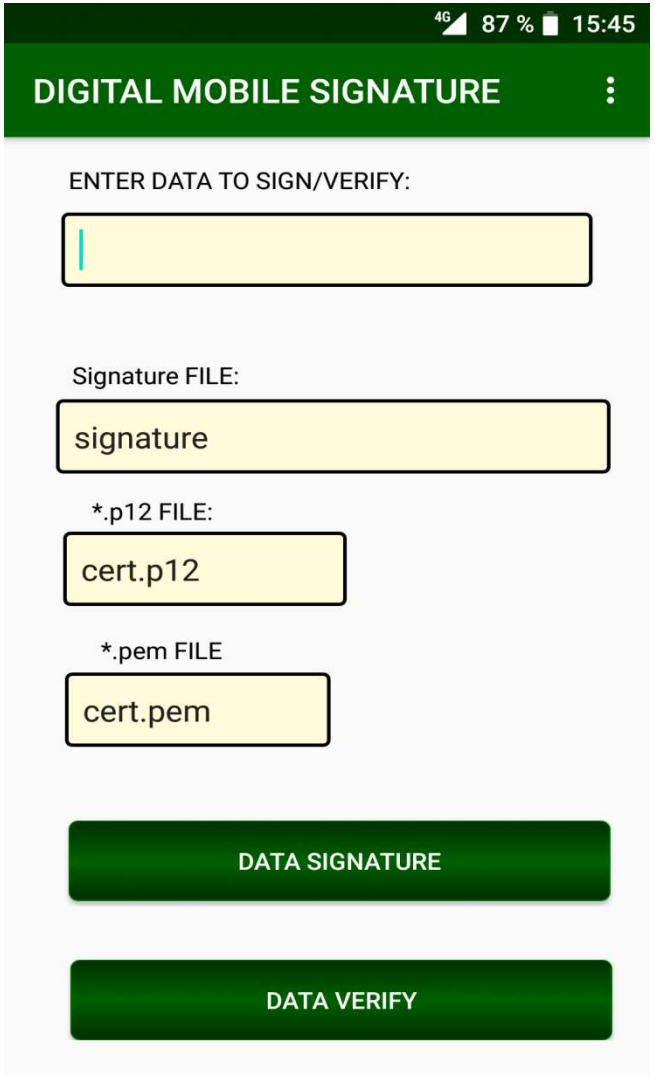


Aplicaciones TLS en Android con ficheros *.p12 y *.pem (II)

```
SSLContext ctx;
 KeyManagerFactory kmf, kmf2;
            //KeyStore  ks2;
kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
            ctx = SSLContext.getInstance("TLS");
 KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
 KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
            InputStream certificateStream =
getResources().openRawResource(R.raw.cacertificado);
CertificateFactory certificateFactory = CertificateFactory.getInstance("X509");
java.security.cert.Certificate[]  chain = {};
 chain = certificateFactory.generateCertificates(certificateStream).toArray(chain);
certificateStream.close();

String Alias="oooooo";
keyStore.load(null,null);
 keyStore.setEntry( Alias,   new KeyStore.TrustedCertificateEntry(chain[0]), null);
TrustManagerFactory tmf =
TrustManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
 tmf.init(keyStore);
 KeyStore ks2 = KeyStore.getInstance(KeyStore.getDefaultType());
 KeyStore keyStore2 = KeyStore.getInstance(KeyStore.getDefaultType());
 InputStream fichp12 = getResources().openRawResource(R.raw.cert);
 String PassStore = "ppppp";
 char[] fraseclave = PassStore.toCharArray();
 ks2 = KeyStore.getInstance("PKCS12");
 ks2.load(fichp12, fraseclave);
 fichp12.close();
 kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());

kmf.init(ks2, fraseclave);

ctx.init(kmf.getKeyManagers(),tmf.getTrustManagers(), null);

SSLSocketFactory sslSf = ctx.getSocketFactory();
SSLSocketFactory sslsocketfactory = ctx.getSocketFactory();
SSLSocket sslsocket = (SSLSocket) sslSf.createSocket(IpClient, ServerPort);
```

```
KeyManagerFactory kmf1;
KeyStore ks1, ks2;

// Load  *.p12 file
String  PassStore = "ppppp";
char[] fraseclave = PassStore.toCharArray();
ks1  = KeyStore.getInstance("PKCS12");
String  myfichero2 = "cert_user.p12";
File  ruta_sd = Environment.getExternalStorageDirectory();
File  fs = new File(ruta_sd.getAbsolutePath(), myfichero2);
InputStream   fichp12= new FileInputStream(fs);
ks1.load(fichp12  , fraseclave);
fichp12.close  ();
kmf1 = KeyManagerFactory.
            getInstance(KeyManagerFactory.getDefaultAlgorithm());
kmf1.init(ks1, fraseclave);

// Load  ca.crt file
ks2 = KeyStore.getInstance(KeyStore.getDefaultType());
ks2.load(null  , "oooooo".toCharArray());
String NameFile1="ca.cer";
File ruta_sd = Environment.getExternalStorageDirectory();
File fs = new File(ruta_sd.getAbsolutePath(), NameFile1);
InputStream certstream= new FileInputStream(fs);
CertificateFactory certFactory = CertificateFactory.getInstance("X509");
java.security.cert.Certificate[]  chain = {};
chain = certFactory generateCertificates(     certstream  ).toArray(chain);
certstream.close();
String Alias="oooooo";
KeyStore.Entry myentry=new KeyStore.TrustedCertificateEntry(chain[0]);
ks2.setEntry ( Alias, myentry , null);
TrustManagerFactory tmf = TrustManagerFactory.
         getInstance(KeyManagerFactory.getDefaultAlgorithm());
tmf.init(ks2);
SSLContext   ctx = SSLContext.getInstance("TLS");
ctx.init(kmf1.getKeyManagers    (), tmf.getTrustManagers(), null);

SSLServerSocketFactory sslserversocketfactory =
                        ctx.getServerSocketFactory();
SSLServerSocket sslserversocket =(SSLServerSocket)
                        sslserversocketfactory.createServerSocket(5223);
……..
```

APP DIGITAL MOBILE SIGNATURE



Código Fuente *Firma* Documentos

```
String data = Text to be signedT";

System.out.println("Creating signature file");
String myfichero = "signature";
File ruta_sd = Environment.getExternalStorageDirectory();
File fs1 = new File(ruta_sd.getAbsolutePath(), myfichero);
FileOutputStream fos = new FileOutputStream(fs1);
ObjectOutputStream oos = new ObjectOutputStream(fos);

//load *.p12 file
String PassStore = "ppppp";
char[] fraseclave = PassStore.toCharArray();
ks = KeyStore.getInstance("PKCS12");
String myfichero2 = "cert.p12";
File fs = new File(ruta_sd.getAbsolutePath(), myfichero2);
InputStream fichp12= new FileInputStream(fs);
// get alias
ks.load(fichp12, fraseclave);
Enumeration<String> enumeration = ks.aliases();
String alias = null;
int n = 1;
while (enumeration.hasMoreElements()) {
      alias = (String) enumeration.nextElement();
       n = n + 1;
}
 fichp12.close();
// get private key
Key key = ks.getKey(alias, fraseclave);
PrivateKey privKey = (PrivateKey) key;
Signature sig = Signature.getInstance("SHA512withRSA", "BC");
 sig.initSign((PrivateKey) key);
byte buf[] = data.getBytes();
//sign document
sig.update(buf);
oos.writeObject(data);
oos.writeObject(sig.sign());
oos.close();
System.out.println("SIGNATURE GENERATED ok!!!!!");
```

Código Fuente
Verificación Documentos

```
// Load  cert.crt  file
CertificateFactory  cf = CertificateFactory.getInstance("X.509");
String myfichero2  = "cert.crt";
File ruta_sd  = Environment.getExternalStorageDirectory();
File fs = new  File(ruta_sd.getAbsolutePath(),  myfichero2);
InputStream caInput = new BufferedInputStream(new
FileInputStream(fs));
Certificate    cert = cf.generateCertificate(caInput);
PublicKey pk= cert.getPublicKey();

//load signature file
File fs2 = new File(ruta_sd.getAbsolutePath(),  myfichero);
FileInputStream fis = new FileInputStream(fs2);
ObjectInputStream  ois = new ObjectInputStream(fis);
Object o = ois.readObject();
 String data = null;
byte signature[]  = null;
//read data signed
data = (String)  o;
//read signature
o = ois.readObject();
signature = (byte[])  o;
ois.close();
fis.close();
System.out.print("DATA:    "+data);

Signature s = Signature.getInstance("SHA512withRSA",   "BC");
s.initVerify(pk);
s.update(data2.getBytes());
 if (s.verify(signature))  {
                System.out.println("Verified  signature  !!!!!");
}
```