Java JCA/JCE programming in Android with SD smart cards

Luis A. Maia, Manuel E. Correia
lmaia@dcc.fc.up.pt , mcc@dcc.fc.up.pt
Center for Research in Advanced Computing Systems (CRACS-INESC LA);
Department of Computer Science, Faculty of Science, University of Porto;
Portugal

Abstract - **The mobile phone is currently the preferred means by which people can communicate and interact with each other at a distance. Not only that, the smart-phone nowadays constitutes the full embodiment of the truly personal device users carry constantly with them, everywhere. They are therefore the ideal means by which the user can casually and conveniently interact with information systems. It can also act as a convenient and highly practical storage place for sensitive identity information.**

Moreover nowadays there is a real urgent need to have in place secure, non-refutable and securely managed identities and communications, with sufficiently strong authentication mechanisms that can assure, among other important properties, strong non-repudiation. In this paper we describe how we have managed to combine the functionalities provided by OpenSC and the University of Graz IAIK java security provider to more easily port java security applications based on the desktop JCA API to the Android platform. This enables us to take advantage of the full extent of the security programming facilities provided by the java programming language on android devices equipped with smart digital (SD) based smart cards.

1 Introduction

With the recent advances in decentralised identity management services, the demand for mobile devices that can manage identities and provide convenient access authorisation is growing. While some work has already been developed to provide platform draft infrastructures and frameworks to build such mechanisms[3],[12], they are all dependent on the security of the keystore provided by the mobile device operating system. This means that a single system failure can result in the compromise of the user private keys and therefore the security of the users identity. When dealing with highly sensitive data, like for example in clinical settings, it is vital to have more trustful user identities and provide a basis for much stronger authentication mechanisms. We believe the more traditional and cheaper operating system file based keystore approach to protect private cryptographic keys of identity certificates is not secure enough for more critical settings where stronger security properties are required. A regular file based keystore can be copied and the smartphone can be a target of attacks where this file can be easily compromised. We think it is reasonable to put the encrypted file based keystore level of security in tandem with the security provided by a much simpler login/password based scheme. In fact an attack on a password protected keystore involves a password guessing attack completely analogous in terms of complexity to what happens with an attack directed towards a login/password scheme, the only thing really different in this case being the need to posses a copy of the keystore file in order to proceed with the attack. To cope with this weakness we can rely on the security properties of SD based smartcards[10] for mobile devices. These security cards are composed by a flash memory and a smartcard component that provides the necessary crypto components and device physical non tampering security features that allows us to reach the level of authentication and non-repudiation required for more security sensitive scenarios. In fact the smart card component of SD flash cards provides the means to solve this problem by maintaining a complete physical separation between the phone operating system and the tamper proof smartcard.

While some work has already been conducted by card vendors[14], a simpler more familiar standard java cryptographic JCE provider for the deployment of SD smart cards in the Android platform has not been available yet. In this paper we describe the architecture and development of such a deployment achieved by employing already developed and well proven libraries and by porting some heretofore unavailable software tools to the Android mobile platform.

In Section 2, we describe the required components and their integration, providing examples of usage by the means

of programming samples in Section 3.

1.1 Smart SD Cards

Smart cards are devices with a single embedded microprocessor chip providing secure data storage and processing. This microprocessor chip is tamper-resistant [11], communicating with the host computer using a protocol interface that enforces control on the data access being logically impossible to extract information without the appropriate keys while providing the ability to run a wide set of cryptographic algorithms on the card itself . The strict control of what can be directly read from the card (even with the appropriate pin) creates a difficulty in forging or copying smart cards unlike some of the previous technologies deployed (ex:magnetic stripe cards).

SmartSD cards are a subset of Advanced Security SD cards which contain a smart card element in the SD Memory of the card and are able to allow the implementation of a secure storage on any mobile phone that supports the ASSD specification. Using this specification allows an host device to select and communicate with a security system in the SD card while maintaining its most basic functionality - Data storage. SD cards are also available in the microSD format, a widespread format included in most mobile phones allowing to bring some of the advantages of smart card element usage to mobile phones.

Figure 1 illustrates the components of a SD card containing an integrated smart card element.

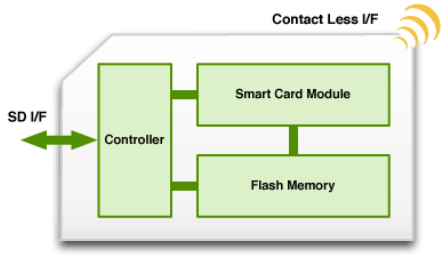

Figure 1: A SD card containing an integrated smart card element.

2 Architecture

In this section we describe the main components of the architecture required to produce a safer mobile identity device with stronger remote authentication capabilities. First and foremost we need to be able to expose an API in the framework that allows us to build applications using SD based smart cards. Since this cards operate with native libraries we demonstrate the steps required to provide support for PC/SC interface in the Android device, and how to use an already available Provider for the Java Platform by convert it to the Dalvik format and cross-compile the native libraries to the platform.

2.1 Mobile Security Cards

A mobile security card (MSC) is a microSD card featuring an integrated smart card element with a JavaCard OS.

Since it was of uttermost interest to select a method which could support vendor independence the obvious choice for communicating with the card was the ASSD specification [1] with the required SmartCard API patches to the android sources. The availability of different devices with different android versions and the need to minimize the impact of forcing an user to install a new Rom, lead to the decision of patching different versions of Android and provide an update file that can be installed from the microSD card for each supported device which contains the patched kernel and software required to communicate with the card. The two devices we have used to test our installation, an LG Etna running a Cyanogen based Rom and an HTC Desire using the latest Android sources available from HTC were updated using a zip file in the microSD card which contained all the necessary changes to the Android Operative System. This zip file contains a set of binaries and scripts to flash the kernel and copy provided files to their respective location in the Android root filesystem.

2.2 Smart Card API

The SmartCard API provides Android Applications the ability to communicate with the SecureElements present by allowing the opening of a dedicated channel to a Secure Element Application and communication using APDU.

The inability to load an applet to a SIM implied a focus on the SD Card as the PKCS#11 device using the provided Java Native Interface [2]for allowing us to communicate with the card inside a Java Application. The Smart Card API is layered as described in the Figure2:

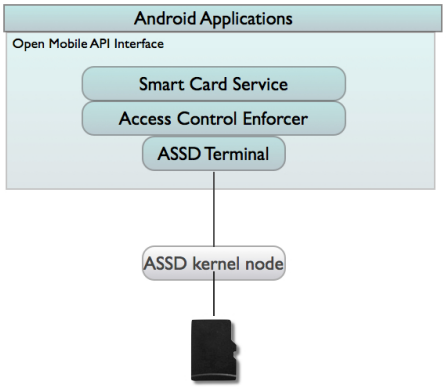

Figure 2: The Smart Card API

2.3 ASSD kernel support

In the ASSD specification, the host is required to use the SD Switch mechanism to query the card in order to verify if the card supports this feature and to set the card in the required mode. Since the ASSD extension has currently two revisions, to detect the version and switch the mode of the card while maintaining compatibility, two different switch codes are in use:

- ASSD 1.1 - 0x1
- ASSD 2.0 (0x4)

In a successful mode switch the host is able to communicate with the card using registers and commands.

2.4 Android Security

The Android security architecture determines that no application by default has permissions to perform operations that could impact other applications, the operating system or the user. This is accomplished with the familiar Unix security provisioning of assigning different userids (UID) and groupids (GUID) to each one of the installed applications[15].

Applications must also statically declare which permissions they may require and the user must give explicit consent during the install. The Smartcard API, takes advantage of this architecture by defining a permission class android.permission.SMARTCARD that the Application must request to obtain access to the API [16]. Since the base of the Android security architecture relies on checking the UID/GID to provide access to a device node, on rooted phones, the super user (root) can access the secure element without any restrictions. But this is also true on any other computer architecture.

2.5 The Android Update Mechanism

Android uses a signed update.zip file stored in external storage as the primary means of releasing and distributing updates to the operating system. We take advantage of this by providing both unsigned and signed (if signing keys are readily available) update files containing an updated kernel to support the ASSD specification, the opensc libraries, and the Java provider and wrapper package.

For already rooted devices we are developing an application to detect the device kernel version so we can detect which kernel and then download the appropriate files and turn the whole process of update and install of the required components more transparent to the user.

2.6 Smart Card Applets

Smart cards evolved from dedicated devices architecture to programmables computing platforms, enabling developers to write card applications and execute them on top of the card virtual machine. A java card is therefore a smart card capable of running java programs, taking advantage of the modularity and encapsulation encouragement brought by the object-oriented programming model of the Java programming language, which relates on more independence between objects and their interaction using well defined interfaces[17]. Due to legacy constraints this cards continue using the APDU (Application Program Data Units) as means of communication between the card and the outside world. The G&D Mobile Security Card is a Java Card which allows to select an Applet to be run on the card's virtual machine. Due to the openness of the Muscle Applet, and because its source is available it is an excellent starting point to development using Java Cards.

To provide the PKI functionality required to the card we built the MuscleApplet[4] from the provided sources and used the G&D JLoad tool to load the applet into the card.

2.7 OpenSC tools and libraries

OpenSC is a set of open source libraries and utilities for accessing smart card management features and cryptographic operations, providing a PKCS#11 interface and supporting numerous smart cards from different vendors as well as the PC/SC interface provided by the pcsc-lite port already available in the SmartCard API patch.

Since OpenSC already supports different Applets and many Java Providers already support OpenSC there is no immediate need to implement the PKCS#11 support using the APDU interface, but leverage instead the availability of the existing implementation[5]. Using the native library opensc-pkcs11.so in the IAIK PKCS#11 wrapper we can use the JCE provider from Java and using the OpenSC layer as displayed in the diagram bellow.

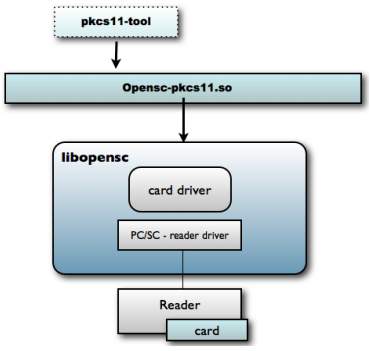

Figure 3: The OpenSC layer

2.8 The IAIK JCA/JCE security provider

We employ the IAIK java cryptographic provider from the Graz University of Technology to be able to use the java JCA/JCE APIs for cryptographic operations within Android[13]. This allows us to more easily port java security programs written in java for the desktop to the Android platform.

It is however important to stress that Android does not employ a regular Java Virtual Machine (JVM) but uses instead, for efficiency reasons, other register based virtual machine called Dalvik VM[2]. This is why the java byte code for the provided by IAIK had first to be converted to Dalvik byte code before being installed on an Android device.

IAIK library implements a JCE provider which provides applications access to the cryptographic hardware through the standard JCA/JCE framework[7]. By providing a simple java.security.Keystore view of the token, to the application developer its as simple as working with file key stores. One key feature is the ability to plug-in our own PIN dialogue, providing the ability to use Android Dialogues without much effort[6].

2.9 The IAIK PKCS#11 wrapper

This is a library which makes the PKCS#11 modules accessible from within Java. Since the PKCS#11 requires native code, we had to cross-compile the source of the native elements using the GCC for ARM EABI and include the modified library in the update zip package we generated.

2.10 Deploying IAIK on Android

We are now able to describe the software layers involved when performing cryptographic operations with IAIK and OpenSC with SD smart cards in Android.

Figure4 illustrates these layers and how they relate and interact with each other.

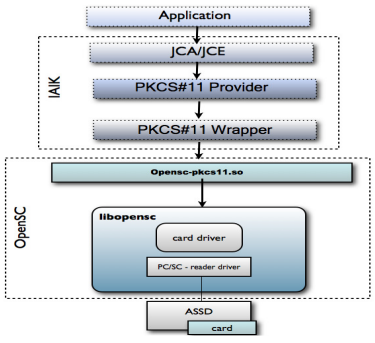

Figure 4: IAIK and OpenSC with SD smart cards in Android

3 Programming with the JCA in Android

3.1 A First Application

Our first application was designed to test the pin request dialog in Android systems using the IAIK JCA provider by implementing the PassphrasePrompt class from IAIK. This class is responsible for displaying a prompt asking the user for the card password/PIN, so since we are using android we should build a Dialog to be able to request the user his password.

```
public class myPassphrasePrompt implements
    PassphrasePrompt{
synchronized public char[] promptPassphrase() {
  AlertDialog.Builder logForm = new AlertDialog.
      Builder(c); // specify the context
  logForm.setTitle("SmartCard Authentication: ");
  logForm.setMessage("Enter Pin:");
  final EditText input =  new EditText(c);
  input.setInputType(InputType.TYPE_CLASS_NUMBER)
      ;
  logForm.setView(input);
  logForm.setPositiveButton("Ok", new
      DialogInterface.OnClickListener() {
   public void onClick(DialogInterface dialog,
       int which) {
    pin=input.getText().toString();
    Log.d("IAIK","IAIK password set to "+pin);
    dialog.dismiss();
    throw new RuntimeException();
   }
  });
  logForm.create().show();
  try{
  Looper.getMainLooper().loop();
  }catch (RuntimeException e){
   return pin.toCharArray();
  }
(...)
```

The following excerpt describes the method used to instantiate a Provider and leveraging the login manager that we implemented.

```
try{
     System.loadLibrary("opensc-pkcs11.so");
     }
     catch (UnsatisfiedLinkError e){
   showDialog(OPENSC_NOT_FOUND);
   return;
     }
     Security.addProvider(new IAIK());
      Properties properties = new Properties();
         properties.put("PKCS11_NATIVE_MODULE", "
             opensc-pkcs11.so");
         Module module = IAIKPkcs11.getModule(
             properties);
          properties.put("SLOT_ID", Long.toString
              (selectedSlot.getSlotID()));
         provider = new IAIKPkcs11(properties);
          /*Specify the pin request with android
              implementation*/
```

```
DefaultLoginManager lm = new
    DefaultLoginManager();
lm.setPassphrasePrompt(new
    myPassphrasePrompt(this));
provider.setLoginManager(lm);

Security.addProvider(provider);
```

The following code excerpt is used to check that the provider is able to correctly log aliases from the keystore. Following the implementation of our Passphrase Prompt, there is no need to modify the code already developed for other JVMs making if effectively portable between platforms without much effort.

```
/*KeyStore*/
    KeyStore cardKeyStore = KeyStore.
        getInstance("PKCS11KeyStore",provider)
        ;
     ByteArrayInputStream
        providerNameInputStream =
     new ByteArrayInputStream(provider.getName
        ().getBytes("UTF-8"));
     cardKeyStore.load(providerNameInputStream
        , null);
     Enumeration<String> en = cardKeyStore.
        aliases();
     while(en.hasMoreElements()){
      Log.d("IAIK","IAIK Token element "+en.
        nextElement().toString());
     }
```

3.2 Digital Signing

To test the signing of some data using the certificates contained in the smart card element we used an adaptation of the following code excerpt:

```
Key key = cardKeyStore.getKey("testkey",null);
PrivateKey signatureKey_ = (PrivateKey) key;
Signature signatureEngine = Signature.getInstance
    ("ExternalSHA1WithRSA",provider.getName());
signatureEngine.initSign(signatureKey_);
signatureEngine.update("This should be signed
    data".getBytes());
byte[] signature_ = signatureEngine.sign();
Log.d("IAIK","IAIK : The signature is:"+new
    BigInteger(1, signature_).toString(16));
```

In order to analyze the overhead of having few API layers and their practical usefulness we took the task to time the execution of the signature block using the lower grade device available. An LG GW620 with 528mhz CPU and with 256 Mb RAM and a computer running windows as a Virtual Machine were used to evaluate the performance of the application performing cryptographic signatures with the results shown on Table 1.

Table 1: Running time in milliseconds

MSC in Android (LG GW620)	CyberFlex 64k (on Windows VM)
2407	880
2176	903
2190	858
2260	877

By analyzing the results we can see that there's some impact in performance when using the MSC in Android.

3.3 Deploying Certs by QRCode

Quick Response codes (QR codes) are two-dimensional square shapes that encode a very reasonable amount of digital information (several Kilobytes of chars) in a small amount of space[9]. The encoding is achieved with the careful positioning of varying size black and white smaller squares within the 2D space defined by the QR square. These 2D codes are normally displayed within web pages or printed in paper posters and are normally employed to quickly exchange reasonable amounts of digital information with mobile devices that would otherwise had to be entered by hand. This is accomplished by having the mobile device to digitally scan and decode the displayed QR code with his built-in optical camera.

In our PKI, QR codes are displayed into the user's desktop display for the auto-enrolment of mobile devices into the PKI infrastructure.

When a signed-in user wants to enroll a mobile device , it request a QR Code containing a symmetric key, generates a key pair in the card and uses the key obtained from the QR Code to provide his public key back to the issuer. The usage of such mechanism means that the provider of the QR Code knows that the device being able to communicate with the symmetric key is owned by the user requesting the QR code, obtains a public key from the device and this without the private key ever leaving the smart card.

QR codes are a very convenient way of conveying a reasonably amount of secret shared information to a mobile device that would otherwise be very cumbersome to input by hand by the user[8]. This usage of QR codes to share secret information between a regular information system and a mobile device equipped with an optical camera can in a way be seen as the establishment of a rather new special security layer that takes advantage of the analog security properties of the optical channel that is employed during the scanning of the QR codes by the mobile device. In other words QR codes can be used in this context to simplify and make practical the auto-enrolment into OFELIA of the users mobile device.

4 Conclusions

We have shown in this paper that is possible to take full advantage of the security provided by the cryptographic functionalities of smart cards on the Android platform using the already well tested and proven facilities provided by the full extent of the java JCA/JCE JDK API. Moreover, by using smart cards to protect PKI certificate keys we are now in a position to strongly authenticate mobile devices on client server networking architectures by having client applications deployed as APPS on android devices that can now have their communications provided by TLS/SSL with client authentication provided by PKI X509 certs whose key are now protected by a real smart card. We demonstrated that by careful installation of OpenSC and the IAIK Graz security provider we can now program this apps, including their TLS/SSL communications, employing the same API (JCA/JCE) as their desktop counterparts, thus making the porting and maintenance of already existing java security software on the Android platform so much familiar and easier to achieve.

Acknowledgments

This work is financed by the ERDF – European Regional Develop- ment Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project PTDC/EIA-EIA/104328/2008".

References

[1] SD Association. Simplified version of advanced security sd specification version 2.0. https://www.sdcard.org/developers/overview/ASSD/ Verified on 14/02/2012. 2

[2] Leonid Batyuk, Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Ahmet Camtepe, and Sahin Albayrak. Developing and Benchmarking Native Linux Applications on Android. In Bonnin, JM and Giannelli, C and Magedanz, T, editor, MOBILE WIRELESS MIDDLEWARE, OPERATING SYSTEMS, AND APPLICATIONS, volume 7 of Lecture Notes of the Institute for Computer Sci- ences Social Informatics and Telecommunications Engineering, pages 381–392, 2009. 2nd International Mobilware Conference, Berlin, GERMANY, APR 28-29, 2009. 2, 4

[3] ATS Chan. Web-enabled smart card for ubiquitous access of patient's medical record. COMPUTER NETWORKS-THE INTERNATIONAL JOURNAL OF COMPUTER AND TELECOMMUNICATIONS NETWORKING, 31(11-16):1591–1598, MAY 17 1999. 1

[4] T Cucinotta and D Corcoran. Muscle cryptographic card edge definition for java enabled smartcards. http://www.musclecard.com/musclecard/files/mcardprot-1.2.1.pdf Verified on 14/02/2012. 3

[5] T Cucinotta, M Di Natale, and D Corcoran. An open middleware for smart cards. COMPUTER SYSTEMS SCIENCE AND ENGINEERING, 20(6):439–450, NOV 2005. 1st International Conference on Trust and Privacy in Digital Business, Zaragoza, SPAIN, AUG 30-SEP 01, 2004. 3

[6] IAIK Graz. Using the iaik jce provider for pkcs#11. http://bit.ly/IAIK_PKCS11 Verified on 14/02/2012. 4

[7] Jonathan Knudsen. Java cryptography. O'Reilly & Associates, Inc., Se- bastopol, CA, USA, 1998. 4

[8] JI Lee, T Kwon, S Song, and J Song. A model for embedding and authorizing digital signatures in printed documents. In Lee, PJ and Lim, CH, editor, IFORMATION SECURITY AND CRYPTOLOGY - ICISC 2002, volume 2587 of LECTURE NOTES IN COMPUTER SCIENCE, pages 465–477. Minist Informat & Commun; Korea Inst Informat Secur & Cryptol, 2002. 5th International Conference on Information Security and Cryptology, SEOUL, SOUTH KOREA, NOV 28-29, 2002. 5

[9] Yue Liu, Ju Yang, and Mingjun Liu. Recognition of qr code with mobile phones. Control and Decision Conference, pages 203–206, 2008. 5

[10] Keith E. Mayes, Konstantinos Markantonakis, Keith Mayes, and Tim Evans. Smart cards for mobile communications. In Smart Cards, Tokens, Security and Applications, pages 85–113. Springer US, 2008. 1

[11] Markus G. Kuhn Oliver Kommerling. Design principles for tamper-resistant smartcard processors. USENIX Workshop on Smartcard Technology, May 1999. 2

[12] Chung-Ming Ou and C. R. Ou. A high-level 3G wireless PKI solution for secure healthcare communications. In Atzeni, AS and Lioy, A, editor, PUBLIC KEY INFRASTRUCTURE, PROCEEDINGS, volume 4043 of LECTURE NOTES IN COMPUTER SCIENCE, pages 254–256. Ist Super Mario Boella, 2006. 3rd European Public Key Infrastructure Workshop (EuroPKI 2006), Turin, ITALY, JUN 19-20, 2006. 1

[13] Karl Scheibelhofer. Using opencard in combination with the java cryptographic architecture for digital signing. In GEMPLUS Developer Conference 2000, pages N/A – N/A, 2000. 4

[14] seek-for android. Pki support on android. http://code.google.com/p/seek-for-android/wiki/SmartCardPKI Verified on 14/02/2012. 1

[15] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google android: A comprehensive security assessment. Security Privacy, IEEE, 8(2):35 –44, march-april 2010. 3

[16] Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, and Shlomi Dolev. Google android: A state-of-the-art review of security mechanisms. CoRR, abs/0912.5101, 2009. 3

[17] Jean-Jacques Vandewalle and Eric Vétillard. Developing smart card-based ap- plications using java card. In Jean-Jacques Quisquater and Bruce Schneier, editors, Smart Card Research and Applications, volume 1820 of Lecture Notes in Computer Science, pages 105–124. Springer Berlin / Heidelberg, 2000. 3