Towards a New Generation of NFC Secure Mobile Services

Pascal Urien
LTCI, UMR 5141
Telecom ParisTech
Paris, France
Pascal.Urien@Telecom-ParisTech.fr

***Abstract*— This paper presents the technical foundations for a new generation of NFC secure mobile services. It introduces trusted mobile applications based on secure elements such as SIM modules or NFC external cards, and uses services built over the *OpenMobileAPI* framework, the *Host Card Emulation* (HCE) environment from Android, and the emerging Remote *APDU Call Secure* (RACS) protocol.**

Keywords—Cloud of Secure Elements; HCE; TLS; Security.

I. INTRODUCTION

Internet technologies increasingly rely on mobile devices. As an illustration, one billion smart phones [1] and more than 300 million tablets [2] were sold in 2013. According to [3], 50 billion connected objects are expected by 2020. In this context, the security of mobile applications is a very critical topic. In this paper, we focus on applications involving Near Field Communication (NFC) interfaces and delivering payments or access control services.

Android is today's dominant operating system in the mobile market. In 2011, the SIMAlliance organization released the OpenMobileAPI [4] that implements a secure framework for SIM access. In November 2011, the NexusS was the first android phone supporting a NFC interface [5]. In October 2013, the Host Card Emulation [6] technology was introduced by the *KitKat* Android version.

A secure element [7] is a trusted microcontroller whose security is enforced by multiple hardware and software countermeasures. This tamper resistant chip is used by various devices such as SIM modules (see Figure 1), EMV payment cards, transportation tickets or electronic passports.

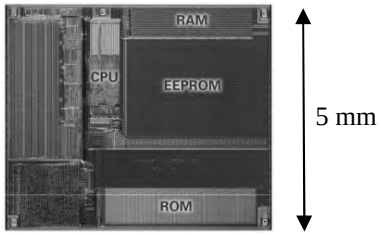

Fig. 1. A SIM secure microcontroller, according to [8].

Furthermore, we recently introduced an open protocol called RACS (Remote APDU Call Secure [9]) whose goal is the remote use of secure elements hosted in the cloud.

In this paper, we analyze how these four technologies (OpenMobileAPI, NFC, Host Card Emulation, and RACS) could collaborate in order to build the foundations of a new generation of secure mobile applications based on Secure Element (SE).

This paper is constructed according to the following outline. Section 2 briefly introduces the state of art for NFC mobile services. Section 3 describes the main OpenMobileAPI functionalities. Section 4 presents the Android NFC interface. Section 5 details the Host Card Emulation facility in Android. Section 6 summarizes the RACS protocol features. Section 7 illustrates the SIM personalization required by the OpenMobileAPI services. Section 8 shows the architecture of the next generation of secure NFC mobile services. Finally, Section 9 concludes this paper.

II. STATE OF ART

The first generation of NFC mobile services [10] is based on NFC-enabled SIMs. Smartphones are equipped with NFC controllers. The SIM chip uses the Single Wire Protocol (SWP, [11]) in order to exchange ISO7816 messages with the NFC controller (see Figure 2).

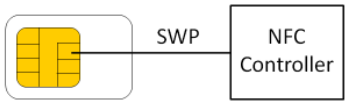

Fig. 2. First generation of NFC Mobile Services

NFC services, for example EMV payment applications, are stored and executed in the SIM module. Usually, there are remotely downloaded from TSM (Trusted Service Manager) servers thanks to the OTA (Over The Air) technology.

This architecture is limited by the three following issues

- The memory size of current SIMs available for application storage (about 100KB)

 ISBN: 978-1-61208-433-6

- The security issues for service providers who do not want to store their applications in non trusted (SIM) devices, owned by mobile operators.

- The functional complexity of OTA and TSM frameworks.

Hopefully these limitations may be overcome by a set of new emerging techniques, which are detailed in the next sections.

III. The Open Mobile API

The OpenMobileAPI [4] is a Java API whose specification is released by the SimAlliance organization (simalliance.org). This framework provides a secure and trusted access for mobile applications in an android context (see [12] for more details). It comprises two packages:

- org.simalliance.openmobileapi

- and org.simalliance.openmobileapi.service.

The first package defines four classes *SEService*, *Reader*, *Session* and *Channel*.

- SEService.class is the abstract representation of all Secure Elements (SE) available for applications running in the mobile phone.

SEService seService = new SEService(this,this)

An instance of SEService is typically created at run time of an Android activity. A callback function defined in the SEService.CallBack interface

public void serviceConnected(SEService service)

is invoked when the service has been successfully launched by the Android operating system. The SEService is shutdown typically when the activity is destroyed according to the procedure

seService.shutdown()

-The Reader class is the logical interface with a Secure Element. It is an abstraction from electronics devices which are needed for contact (ISO 7816) and contactless (ISO 14443) smartcards. The procedure

Reader[] readers = seService.getReaders()

returns an array of available Readers. Generally, only one reader associated to a SIM card is available in a smartphone

-The Session class opens and closes a session with an embedded Reader thanks to methods

Session session = readers[0].openSession()

and session.close() or readers[0].closeSessions()

It establishes the logical path with the Secure Element managed by the Reader.

- The Channel class object is associated with an application identified by an application identifier (AID, a byte array of 16 elements at the most) and running in the Secure Element. The procedure,

Channel channel = session.openLogicalChannel(aid)

starts an application in the SIM card, according to a communication framework called logical channel by the ISO7816 standards. The method

channel.close(), stops a channel previously opened

Channels objects are used to send ISO7816 requests and to receive ISO7816 responses to/from SIM cards. These operations are performed thanks to the method :

byte[] response channel.transmit(byte[] command)

IV. The NFC Interface

The Android operating system supports NFC (Near Field Communication) interface since the 2.3 (Gingerbread) version [5]. Because NFC devices usually embed a secure element, this inductive coupling interface is another way to use a SE from a mobile application (see, for example, [13]).

Applications must be registered for android.permission.NFC events. When the operating system detects an external NFC card, it sends an INTENT to registered applications. The NfcManager class enumerates the NFC adapters available on the Android device board. Usually there is only one chip, so the static method getDefaultAdapter() returns an instance the class NfcAdapter, which represents the hardware NFC controller.

The NfcAdapter.ACTION_TAG_DISCOVERED INTENT notifies the detection of an external NFC tag. Thereafter, a Tag object is retrieved from the INTENT, which is afterwards converted to an IsoDep object (named Dev in figure 3). As illustrated in Figure 3, it is possible to exchange ISO7816 requests and responses thanks to the *transceive* procedure from the IsoDep class,

byte [] response Dev. transceive (byte[] resquest)

```
private void resolveIntent (Intent intent)
throws IllegalArgumentException, IllegalAccessException  {

String action = intent.getAction();
 if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action))
 { Tag tag = intent.getParcelableExtra
  IsoDep Dev = IsoDep.get(tag) ;
   if (TagI== null) return ;

  try { Dev.connect();}
  catch (IOException e) {return;}

  byte request[]= {(byte)0x00,(byte)0xA4,(byte)0x04,(byte)0x00,
                   (byte)0x07,(byte)0xA0,(byte)0x00,(byte)0x00,
                   (byte)0x00,(byte)0x30, (byte)0x00,(byte)0x01};

// Send ISO7816 Request, Receive Response
try { byte[] response= Dev.transceive(request);
catch (IOException e) {return;}

   try {Dev.close();}
   catch (IOException e) {return;}

} }
```

Fig. 3. Android NFC Interface

V. HOST CARD EMULATION

Host Card Emulation (HCE, [6]) has been introduced since the version 4.4 (KitKat) of the android system.

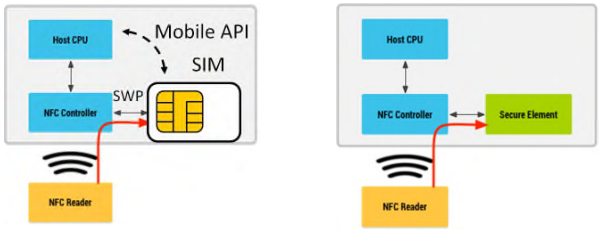

Fig. 4. SWP SIM centric architecture (left part) and the Nexus S Secure Element embedded in a NFC controller (right part)

Before Android 4.4, some devices, such as the NexusS, worked with NFC controllers embedding Secure Elements, while other devices could use SIM card with SWP (Single Wire Protocol) pad exchanging ISO7816 messages with the NFC controller (see Figure 4).

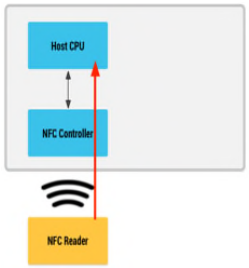

Fig. 5. The android KitKat Host Card Emulation

What is new with the HCE paradigm is that the operating system may have an exclusive control over the NFC controller (see Figure 5).

Applications intending to use HCE services must be registered to the android.permission.BIND_NFC_SERVICE permission, as illustrated in Figure 6. They must include a dedicated service (named MyHostApduService in Figure 6) extending from the *HostApduService* class. A specific xml file (named apduservice.xml in Figure 7) gives a list of supported application identifier (AID), as shown in Figure 7.

```
<service
 android:name=".MyHostApduService"
  android:exported="true"
  android:permission="android.permission.BIND_NFC_SERVICE" >
 <intent-filter>
<action android:name=
"android.nfc.cardemulation.action.HOST_APDU_SERVICE" />
 </intent-filter>

<meta-data
  android:name="android.nfc.cardemulation.host_apdu_service"
  android:resource="@xml/apduservice" />
</service>
```

Fig. 6. The Host APDU service in Android KitKat

```
<host-apdu-service
  xmlns:android= "http://schemas.android.com/apk/res/android"
  android:description="@string/servicedesc"
  android:requireDeviceUnlock="false" >

<aid-group
  android:category="other"
   android:description="@string/aiddescription" >
   <aid-filter android:name= "325041592E5359532E4444463031" />
   <aid-filter android:name= "a0000000041010aa54303200ff01ffff" />
</aid-group>

</host-apdu-service>
```

Fig. 7. Host Card Emulation service desrcription in Android KitKat

The HCE service implements two methods for NFC communication:

-public byte[] processCommandApdu(byte[] apdu, Bundle extras). This procedure received ISO7816 requests transmitted by the NFC reader.

- public void sendResponseApdu(byte[] responseAPDU). This procedure forwards ISO7816 requests to the NFC reader.

In summary, applications registered for particular AIDs implement dedicated services that process ISO7816 requests and produce ISO7816 responses.

VI. REMOTE APDU CALL SECURE RACS

RACS is an open protocol based on an IETF draft [9]. The goal of RACS is the remote use of secure elements hosted in dedicated servers. RACS works over TLS, which is the de facto standard for secure data exchange over the Internet.

RACS is based on a TLS/TCP/IP stack (see [14] for more details) and provides two main services:

- The inventory of the Secure Elements hosted by the RACS server

- The transport of ISO7816 requests and responses. Each secure element is identified by a SEID (*Secure Element Identifier*).

A SEID is a name deduced from a secure element pseudo unique identifier, a reader serial number or a physical slot number (see [14]).

Therefore, a secure element in a RACS ecosystem is associated to a Uniform Resource Identifier (URI) such as

Server:Port/SEID

composed of three parts, the RACS server name, the TCP port and the SEID.

Both client and server are equipped with X509 certificates and their associated private keys; a strong mutual authentication is performed between these two entities over a TLS session.

Since RACS clients are authenticated by their certificates, the server handles a security policy in order to

 ISBN: 978-1-61208-433-6

enforce access control to secure elements and their embedded applications identified by AIDs. Figure 8 summarizes the list of RACS available commands.

Command	Purpose
GET-VERSION	Get the current RACS protocol version
SET-VERSION	Set the working RACS protocol version
LIST	Get the list of available and authorized secure elements
POWERON SEID	Power on a secure element
RESET SEID	Reset a secure element
SHUTDOWN SEID	Power off a secure element
APDU SEID	Send an ISO7816 request and return the ISO7816 response

Fig. 8. The RACS commands

VII. The SIM Module Personnalization

For security reasons, the OpenMobileAPI framework manages [4] a security policy based on the hash (SHA1) of a certificate identifying an authority that is granted access, for example a CA certificate used to sign an Android application.

The SIM Master File (MF), the equivalent of a secure element root directory stores the EF-DIR file which contains a reference to PKCS#15 environment comprising either a PKCS#15 SIM application, a PKCS#15 repertory (DF-PKCS#15) name or both. In Figure 9 the name of the DF-PKCS#15 repertory is 7F50.

The DF-PKCS#15 repertory contains at least the mandatory file ODF (Object Directory File), whose default name is 5031. This file holds a reference to the Data Object Directory File (DODF), whose value is 5207 in Figure 9.

DODF stores the name of Access Control Main File (EF-ACMain, 4200 in Figure 9), which contents is the reference of the Access Control Rules File (EF-ACRules, 4300 in Figure 9).

EF-ACRules (see figure 10) contains a list of Application identifiers (AID) and their associated Access Control Conditions File (EF-ACCondition).

Each EF-ACCondition stores access control conditions expressed as a list of entries, each entry containing a SHA-1 of a certificate identifying an authority that is granted access.

- If this file is empty, it means that rules pointing to this file are denying access to any terminal application.

- If this file contains a condition without a certificate hash, then rules pointing to this file are granting access to any terminal application.

```
MF (3F00)
|-EF-DIR (2F00) -->  reference to DF-PKCS#15
|
|-DF-PKCS Access Control Main File #15 (7F50)
  |-ODF (5031)          -->  reference to DODF
  |-DODF (5207)         -->  reference to EF-ACMain
  |-EF-ACMain (4200)    -->  reference to EF-ACRules
  |-EF-ACRules (4300)   -->  reference to EF-ACConditions
  |-EF-ACConditions1 (4310)
  |-EF-ACConditions2 (4311)
  |-EF-ACConditions3 (4312)
```

Fig. 9. SIM files required by the OpenMobileAPI

```
30 10
  A0 08 // aid
    04 06
      A0 00 00 01 51 01 // Application Identifier (AID)
        30 04
          04 02
            43 10 // EF-ACCondition  File
30 10 A0 08 04 06 A0 00 00 01 51 02 30 04 04 02 43 11
30 10 A0 08 04 06 A0 00 00 01 51 03 30 04 04 02 43 11
30 08
  82 00 // other
    30 04
      04 02
        43 12 // file
FF FF FF 90 00
```

Fig. 10. The Access Control Rules File (EF-ACRules)

VIII. New Generation of Secure NFC Mobile Application

Legacy NFC applications work with SIM modules equipped with SWP (*Single Wire Protocol*) pad, and communicating through this link with the NFC controller. The SIM provides services such as payment or access control that are loaded and executed in this secure chip. One issue is the non volatile memory size, typically around 100KB. Another issue is the complexity of the remote administration of the SIM content according to the OTA (Over The Air) technology. We are currently developing (see [15] for example) a new model for NFC applications, which is based on the following components:

- A mobile supporting the Host Card Emulation (HCE) mode.

- A secure element, either a SIM module or an external NFC card, which runs a TLS client stack. The *OpenMobileAPI* or the NFC-API are in charge of the secure element logical access.

- An application that realizes the logical glue between HCE, the TLS stack running in the internal SE, and external SEs stored in a RACS server.

In a simple use case, such as payment operation (see Figure 11), the merchant terminal exchanges ISO7816 requests and responses with the mobile HCE. The mobile application opens a TLS session with a RACS server hosting

 ISBN: 978-1-61208-433-6

payment chips. Thereafter, the application transparently relays ISO7816 messages between the payment terminal and the selected payment chip hosted in the RACS server. For performances issues, the application may handle ISO7816 reading operations (in a cache), but cryptographic procedures (such as CDA, DDA, ARQC...) must be executed by a SE plugged in the RACS server.

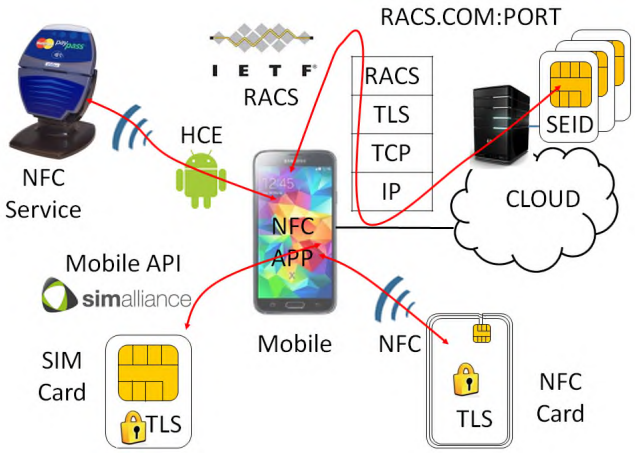

Fig. 11. New generation of mobile secure NFC applications

We detailed in [15] the integration of TLS stack (and the associated performances) in secure elements (see Figure 12), such as SIM cards or NFC devices. Thanks to this technology, the TLS session is booted from the secure element that fully manages a strong mutual authentication based on PKI asymmetric mechanisms. For mobile operators, the SIM acts as an identity module (including an X509 certificate), which gives access to secure elements hosted in the Cloud. The resulting benefits are in consequence the following:

- No limitations induced by SIM memory size for NFC applications storage

- A simple management model for secure elements hosted in the Cloud

- A highly trusted architecture, based on physical isolation properties, because NFC applications are not running in the SIM.

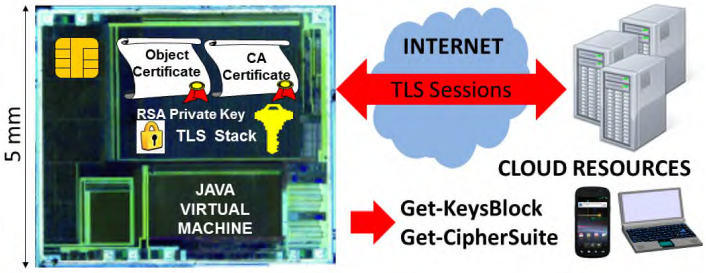

Fig. 12. TLS stack for Secure Elements

Nevertheless, a next generation of RACS servers could be based on Hardware Secure Module (HSM) providing secure elements emulation, but offering ISO7816 interfaces.

IX. CONCLUSION

In this paper, we presented the foundations for a new generation of secure mobile NFC services. We are currently working on the design of an open platform, dedicated to payment, as illustrated in Figure 13.

Fig. 13. RACS open platform for payment applications according to [17].

REFERENCES

[1] http://www.idc.com/getdoc.jsp?containerId=prUS24645514, last access June 2015

[2] http://www.gartner.com/newsroom/id/267421, last accesss June 2015

[3] D. Evans, "The Internet of Thing. How the Next Evolution of the Internet Is Changing Everything", Cisco White Paper, 2011

[4] GSMA, "Mobile NF SIM Alliance, "Open Mobile API specification V2.02", 2011

[5] Near Field Communication, https://developer.android.com/guide/topics/connectivity/nfc/index.html, last access June 2015

[6] Host Card Emulation, https://developer.android.com/guide/topics/connectivity/nfc/hce.html, last access June 2015

[7] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO)

[8] Klaus Vedder, "Smart Cards", ETSI Security Workshop 2006, http://www.etsi.org/WebSite/document/Workshop/Security2006/Security2006S1_3_Klaus_Vedder.pdf, June 2015

[9] Remote APDU Call Secure (RACS), draft-urien-core-racs-03.txt, IETF draft August 2014

[10] GSMA, Mobile NFC Technical Guidelines, Version 2.0, November 2007

[11] ETSI TS 102 613 V7.3.0 (2008-09), Technical Specification Smart Cards; UICC - Contactless Front-end (CLF) Interface; 2008

[12] Urien,P., "An OPENID Identity Service for Android, Based on USIM Secure Elements", in proceedings of MobiCASE 2012, Seattle, Washington, USA, 11-12 October 2012

[13] Urien, P., Kiennert, C., "A New Key Delivering Platform Based on NFC Enabled Android Phone and Dual Interfaces EAP-TLS Contactless Smartcards", in proceedings of MobiCASE 2011, Los Angeles, CA, USA, October 24-27, 2011.ISBN 978-3-642-32319-5

[14] Urien, P.; "RACS: Remote APDU Call Secure Creating Trust for the Internet", The 2015 International Conference on Collaboration Technologies and Systems, Atlanta, Georgia, USA, CTS 2015

[15] Urien, P.; "Cloud of Secure Elements: An Infrastructure For the Trust of Mobiles NFC Services", The 10th IEEE WiMob 2014, October 8-10 2014, Larnaca, Cyprus

[16] Urien, P., Betirac, M., "A Triple Interfaces Secure Token -TIST- for Identity and Access Control in the Internet Of Things", SMART 2013, June 23 - 28, 2013 - Roma, Italy

[17] HCE-SIM, Payment with an open RACS platform https://www.youtube.com/watch?v=DpTAyDNIVLc, last access June 2015

 ISBN: 978-1-61208-433-6