Journal of Mobile, Embedded and Distributed Systems, vol. I, no. 1, 2009
ISSN 2067 – 4074

Security Issues for JME M-Applications

Catalin BOJA

Faculty of Cybernetics, Statistics and Economic Informatics
Department of IT&C Technologies
Academy of Economic Studies Bucharest, Romania
catalin.boja@ie.ase.ro

***Abstract**:* The paper describes the concept of data security in a mobile environment. The objective is to develop Java software applications, MIDlets for mobile devices, which implement data protection at different levels. Existing and proposed solutions are described by defining security models and architectures. From the software development point of view, the paper describes two API's, JSR 177 [1] and Bouncy Castle Crypto APIs [2] that provide methods to reach needed security levels. The paper is a survey of security practices for J2ME software platform describing the goals and the means to reach them.

***Key-Words**:* M-applications, JME platform, mobile architectures.

1. Introduction

The exponential growth of mobile devices sales, over 1.2 billion phones sold in 2008, from which 14% are smartphones, [7], represents the premise of an equal growth in the development of mobile applications. By default, devices come with a wide range of software programs, for personal information management, but the platform allows other software developers to provide additional products. The increase need to be informed or to stay connected with different data services requires a mobile platform, both hardware and software, that can offer secure and reliable data services when and where are needed.

Accordingly to [8], 80% of handsets now support Java ME and in 2007 there were 3 billion Java powered devices worldwide. Thus, a large number of mobile applications are intended for the JME platform, making it a large market for software development and also a vulnerable target from the viewpoint of data security.

The mobile technologies are one of the most successful technologies that have emerged into society. One reason for that is the relative short time in which it has evolved to a level that it can provide many tasks you can perform with a desktop computer. Other reasons are based on the advantages of using small mobile devices that allow you to send and receive data with a relative ease from different locations.

The emergence of data storing and processing solutions on mobile devices has resulted from an increased mobility of users and the advancement in wireless technologies. This has allowed the appearance and fast development of new solutions in domains like education (m-learing), health and care services (m-health), commerce and other public services.

Mobile devices make data access available for applications and users by:

- wireless network connections using GPRS or 3G data services provided by the network carrier;
- wireless networks based on 801.11g protocol;
- data cables that allow connections between personal computers and mobile devices; different data synchronization services are available;
- mobile browsers based on WAP (Wireless Application Protocol) and WML (Wireless Markup Language);

ISSN 2067 – 4074

- infrared ports;
- bluetooth short range wireless connections.

Mobile technologies allow users to store large amounts of data on the mobile device. Data storage is done using internal memory of the device, memory cards with capacity up to multiple GB and remote storage using network access connections.

2. Security API for JME applications

Since its first release in 1999 Java Micro Edition platform is intended for all mobile devices that have limited hardware and software resources. In order to allow the development of application for these devices, this platform has limited framework support comparing to its big brother, Java Standard Development Kit.

Despite the early characteristics of devices, present and future mobile technologies allow software developers to design and implement secure solutions for storing and accessing data. In order to support this kind of resources, Java ME platform includes a package, the Security and Trust Services API (SATSA), [1], that is flexible enough to run with many types of cryptographic algorithms and protocols. The SATSA framework has been designed to run on any J2ME-based virtual machine, including the CDC and CLDC configurations. This Java standard specification has been defined by the Java Community Process (JCP) in JSR 177, [1].

The SATSA package is described in table 1, [1], [9-10].

Table 1. Overview of the SATSA package architecture.

Package Name	Implementation	Processing Unit
SATSA-APDU Communication API	Defines the `APDUConnection` interface allowing MIDlets to communicate with the smart card	Smart card
SATSA-JCRMI Communication API	Allows MIDlets to requests method invocation to smart cards	Smart card
SATSA-PKI Signature Service API	Allows MIDlets to requests cryptographic signature or authentication to smart cards	Smart card
SATSA-CRYPTO Cryptographic API	Allows MIDlets to implements and use cryptographic methods	Mobile device
Generic Connection Framework	Genneric connector for JCRMI and APDU connections	Smart card

The API provides interfaces that allow developers to implement secure solutions based on a smart card, the mobile device or a combination of the two. This survey concentrates only on the second solution, using only the mobile device processing unit, because there are other restrictions, legal and technical, that will not allow a smart card solution intended for a wide range of devices.

From all the SATSA packages, the one that does not require a smart card is the SATSA-CRYPTO package. It provides classes for implementing data security architectures based on message digests, digital signatures and symmetric and asymmetric encryption / decryption algorithms. A short description of the package is provided in table 2, [1], [9], [10].

ISSN 2067 – 4074

Table 2. SATSTA-CRYPTO package.

Package	Classes and Interfaces	Description
`java.security`	Key PublicKey KeyFactory MessageDigest Signature	Public key encryption using RSA, [11], asymmetric algorithm; SHA-1, [11] message digest computation; SHA1withRSA digital signature;
`java.security.spec`	EncodedKeySpec X509EncodedKeySpec AlgorithmParameterSpec	Key specifications and algorithm parameter specifications;
`javax.crypto`	`Cipher`	Data symmetric encryption / decryption using DES, [11], DES-EDE and AES algorithms in ECB or CBC modes;
`javax.crypto.spec`	IvParameterSpec **SecretKeySpec**	Initialization vectors for DES in CBC mode and RSA ciphers Key specifications for DES or Triple DES

In [11] there are described all the symmetric and asymmetric, pubic key, encryption algorithms implemented in the SATSTA-CRYPTO package.

In parallel with the development and release of the SATSA standard there are open projects that offer alternative support for implementing security architectures in a mobile application. One of these solutions is the Bouncy Castle Crypto APIs, [2], that provides a lightweight cryptography API and a provider for the Java Cryptography Extension and the Java Cryptography Architecture. This API can be used and implemented in applications for the J2ME platform or for the JDK 1.6.

Table 3. Overview of the Bouncy Castle Crypto APIs package architecture.

Package Name	Implementation	Processing Unit
net.sourceforge.jcetaglib.lib	Allows MIDlets to implements and use cryptographic methods	Mobile device
net.sourceforge.jcetaglib.taglib.crypto	Allows MIDlets to implements and use cryptographic methods (symmetric and asymmetric encryption; message digest)	Mobile device
net.sourceforge.jcetaglib.taglib.x509	Allows MIDlets to manage signatures	Mobile device
other packages	Additional packages for exceptions, tests, tools	Mobile device

3. Local Data Security

In a mobile environment it is presumed that the application provides users with data without requiring a persistent data connection. Data is downloaded once on the device and used by applications, most of the time, in an offline scenario. That means data is stored on the device and the application provides the interface between it and user needs.

On the device, possible locations where data can be stored are:

ISSN 2067 – 4074

- active memory; mobile devices have a few megabytes of volatile memory; storing data in this zone for a long period of time affects the performance of the application and the mobile device; also, this data is not persistent and an interruption of the application will erase it;
- small databases that run as services in the background; on the Windows Mobile platform, applications can use a lightweight version of a SGBD, Sql Server CE; on the J2ME platform there are no default solutions because these are Running processor intensive applications that affects the performance of the device and implicitly the rate at which battery needs to be recharged;
- persistent memory; in order to allow applications to store data between working sessions and when the device is closed, J2ME platform provides a simple record-based persistent storage mechanism known as the RMS (Record Management System); the RMS system can be viewed as a very small database that stores binary data within a record store allowing mobile applications to add, remove or update data; initially the J2ME specifications does not provided an API for accessing local files because it represents a break in the MIDP sandbox security model; there are many situations that requires large sized resources, like multimedia applications, that RMS can't manage; the possibility to extend device external memory by using memory cards offers the only solution to those situations;

A common scenario for a mobile application that needs to process local stored data is the log-in component. This routine checks user credentials, username and password, in order to authenticate and give access to a user. The user input values must be compared with the user accounts data which is in the device persistent memory.
MIDlet applications can o store persistent data, with the RMS system, figure 1, within a controlled environment, while maintaining a basic system security. RMS offers security by the way it is implemented. Each application defines its RMS space by using a unique ID. Because the data location within the device memory is dependent on the RMS ID and is not exposed to an inquiring MIDlet, data is secured, if the ID value is keep secret and is known only by the application.

```
RecordStore rs;
private static final String ID_RMS = "MyRMS";
 try{
        rs = RecordStore.openRecordStore(ID_RMS, true);
 }
 catch(Exception err){
…
}
```

The previous code sequence it is used to open a RMS record store using an ID.

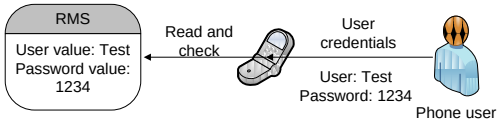

Fig. 1 Storing clear data within RMS

While this scenario offers a relative data security it is vulnerable to reverse engineering and brute force attacks on device memory. Once the RMS ID is known then any other application can access the data.

ISSN 2067 – 4074

In order to increase the level of security for local stored data, there are implemented techniques to hide clear data behind a cipher value. The cipher can be obtained by computing a message digest value, figure 2, or by encryption, figure 3.

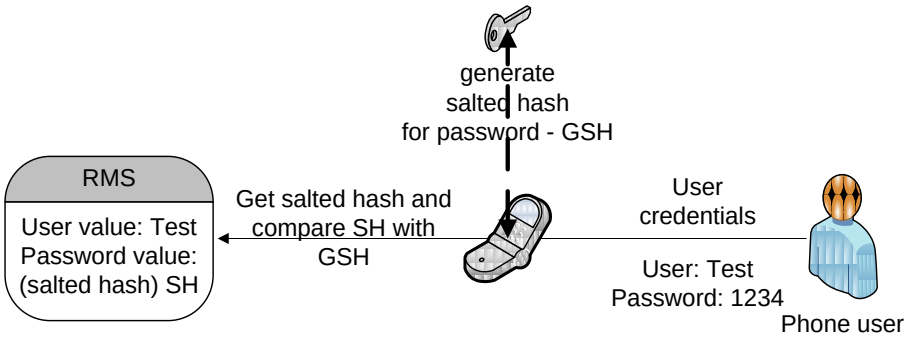

Fig. 2 Storing password salted hash within RMS

If clear data represent short size sensitive values, like passwords or access keys, it can be stored using message digest values, figure 2. For each value it is determined a hash value using a message digest algorithm, [14], and the result is stored in RMS. As message digest functions are one-way it is not possible to obtain the sensitive value from its hash value.

Message digest functions do not provide a perfect security solutions because they are prone to collisions and someone who knows the hash could try to find another value that generates same message digest. Also, a brute force or dictionary attack can be used to guess the password. To protect the hash value from being used to obtain the input value, sensitive data is salted before computing its hash value. The salt is an additional value added to the clear password in order to modify the message digest.

The next sequence generates a SHA1 160 bits, 20 bytes, hash value for a string of data, which can be the salted form of the password. The J2ME platform also supports MD5 message digest, which generates a 128 bit hash value, but it is recommended to use the more secure SHA1.

```
byte[] saltedPassword = "...";
byte[] digestValue = new byte[20];
try {
   MessageDigest sha;
   sha = MessageDigest.getInstance("SHA-1");
   sha.update(saltedPassword, 0, saltedPassword.length);
   sha.digest(digestValue, 0, 20);
} catch (Exception e) {
   // Handle NoSuchAlgorithmException or DigestException
   ...
}
```

An alternative to the message digest solution is the use of encryption algorithms. This approach generates by encryption a cipher text from a clear text. In order to obtained clear data, the reverse process is applied decrypting the cipher. The cipher can be stored using RMS or the file system.

ISSN 2067 – 4074

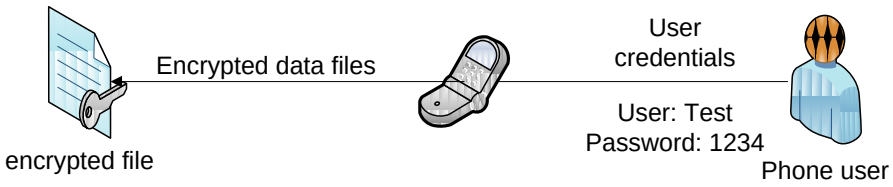

Fig. 3 Storing encrypted password and data in local files

The drawback of encryption is that its implementation is an intensive-processor routine and that affects the application performance and the battery cycle. As described in table 2 and table 3, the two API's offer supports for various symmetric and asymmetric encryption algorithms.

The next sequence encrypts 8 byte clear text using DES symmetric algorithm and the CBC (Cipher Block Chaining) mode, [14].

```
byte[] secretKey = "..."; // secret key
byte[] iv = "..."; // initialization vector
byte[] plainText = "..."; // 8 bytes plain text

try {
   Key key = new SecretKeySpec(secretKey, 0, secretKey.length, "DES");
   Cipher cipher;
   cipher = Cipher.getInstance("DES/CBC/NoPadding");
   if (iv == null) {
      cipher.init(Cipher. ENCRYPT_MODE, key);
   } else {
      IvParameterSpec ivps;
      ivps = new IvParameterSpec(iv, 0, iv.length);
      cipher.init(Cipher.ENCRYPT_MODE, key, ivps);
   }
   int ciphertextLength = 8;
   byte[] cipherText = new byte[ciphertextLength];
   cipher.doFinal(plainText, 0, plainText.length, cipherText, 0);
} catch (Exception e) {
        ...
}
```

A practical implementation of this model is the DRM (Digital Rights Management) standard that protects multimedia resources from being mass distributed without acquiring the rights to use them. A detailed description of the DRM architecture is described in [12].

4. Conclusions

A high level for data security is a goal with equal importance as quality, reliability and performance in the development process of a mobile application. For now, only applications that process sensitive data, as governmental, military and banking, are implementing security components, because that is one of the most important specifications.

In other types of mobile applications, as personal information managers, the security modules are considered optional because they have a negative impact on the process

ISSN 2067 – 4074

performance or are considered not necessary. As mobile devices will store more information, personal and business related, and will gain an increase processing power, the concern of securing the data must be an omnipresent objective in the development lifecycle.

As more API's, proprietary and open source, are becoming available for developers, there are needed comparative analysis to evaluate their quality and more important, their efficiency. In the end, the cryptographic algorithms are the same, the difference is given by the way the security API is implementing them.

References

[1] Java Community Process – *JSR-177 - Security and Trust Services API for J2ME*, http://jcp.org/en/jsr/detail?id=177
[2] *** – Bouncy Castle Crypto APIs, http://www.bouncycastle.org/
[3] Jonathan KNUDSEN – *Wireless Java—Developing with J2ME*, Second Edition, APRESS, 2003, ISBN 1-59059-077-5
[4] John MUCHOW – *Navigate the file system on a mobile device*, 2005, http://www.ibm.com/developerworks/edu/wi-dw-wi-navigate-i.html
[5] Catalin BOJA – *Data Security Solution for Mobile Applications*, The 9th International Conference on Informatics in Economy IE 2009 - Education, Research & Business Technologies, Bucharest, May 7-8 2009.
[6] Michael Howard, David LeBlanc – *Writing Secure Code*, Second Edition, Microsoft Press, 2003, ISBN 0-7356-1722-8
[7] Gartner – *PC Vendors Eyeing Booming Smartphone Market* and *Worldwide Mobile Phone Sales Grew 6 Per Cent in 2008, But Sales Declined 5 Per Cent in the Fourth Quarter*, www.gartner.com, http://www.gartner.com/it/page.jsp?id=904729
[8] *** – *The official Java website*, www.sun.java.com
[9] C. Enrique ORTIZ **–** *The Security and Trust Services API for J2ME*, Sun Developer Network (SDN), 2005, http://developers.sun.com/mobility/apis/articles/satsa1/
[10] *** – *SATSA Developer's Guide*, SATSA Reference Implementation 1.0, December 2004, http://java.sun.com/j2me/docs/satsa-dg/
[11] Alfred J. MENEZES, Paul C. van OORSCHOT, Scott A. VANSTONE - Handbook of Applied Cryptography, CRC Press,1997 (http://www.cacr.math.uwaterloo.ca/hac/)
[12] Ion IVAN, Cristian TOMA, Marius POPA, Cătălin BOJA – *Secure Architecture for the Digital Rights Management of the M-Content*, Proceedings of the WSEAS International Conferences, 5th WSEAS Int. Conf. on Information Security and Privacy, ISP '06, Venice, Italy, November 20 – 22, 2006, CD ISSN 1790 – 5095, ISSN 1790 – 5117, ISBN 960-8457-56-4.
[13] Michael YUAN, Ju LONG - Securing wireless J2ME, *Security challenges and solutions for mobile commerce applications,* IBM Technical library, 2002, http://www.ibm.com/developerworks/wireless/library/wi-secj2me.html
[14] Cristian Toma – *Security in software distributed platforms*, ASE Printing House, Bucharest, 2008, ISBN 978-606-505-125-6.
[15] Victor Valeriu PATRICIU, Ion BICA, Monica Ene-PIETROSEANU, O. PRIESCU – *Semnatura electronica si securitatea informatica*, Publishing House All, Bucharest, 2005.
[16] *** – *The official Bluetooth website*, http://www.bluetooth.com
[17] Martin de JODE – *Programming Java 2 Micro Edition on Symbian OS, A developer's guide to MIDP 2.0*, Wiley, 2004, ISBN 0-470-09223-8