Secure Extended Memory for Java Cards

Serge Chaumette, Achraf Karray*, and Damien Sauveron**

LaBRI, UMR CNRS 5800 – Université Bordeaux 1
351 cours de la Libération, 33405 Talence CEDEX, FRANCE.
{serge.chaumette,achraf.karray,damien.sauveron}@labri.fr

Abstract. Even if the memory size of smart cards has grown significantly, it is still too limited to run real life complex applications. To overcome this limitation we have developed an off-card secure storage solution for Java Card™[1] applications. It is the topic of this paper.

Keywords: *Smart Cards, Java Cards, Secure Storage, Off-card Resources.*

1 Introduction

Even if in the next two years smart cards will embed 1GB of memory, a solution to manage large amounts of data is needed today. To overcome the limited memory size of the smart cards implementing the Java Card technology[2], we propose a solution that makes it possible to store data off-card, still ensuring their confidentiality and their integrity, so as to be in a perfect adequacy with the security requirements which characterize the smart card.

2 Secure Memory Extension

Our proposed solution offers Java Cards a secondary storage (the hard disk of a host) which increases the memory capacity for the embedded applets: for short lived data in a way similar to a virtual memory mechanism (as can be found in most operating systems such as Windows, Linux, etc.); for long lived data as with secure data repositories.

To implement this feature, a dedicated applet running on the Java Card, called the *Secure Storage Manager Unit* (SSMU), ensures the integrity of the swapped data and their confidentiality by ciphering them, and a dedicated application on the host writes the ciphered data sent by the SSMU, manages the identification of ciphered data records and returns back data to the card on demand.

In the process illustrated figure 1, a client applet which needs more storage space, calls, through the firewall provided by the Java Card Runtime Environment, a method of a shareable interface provided by the SSMU applet, to pass it the data to save. When the client applet needs its data back, it calls an other method of the interface provided by the SSMU applet.

---

* LaBRI (Bordeaux, FRANCE) and University of Sfax, ENIS, TUNISIA

** XLIM (Limoges, FRANCE).

[1] Java and all Java-based marks are trademarks or registered trademarks of Sun microsystems, Inc. in the United States and other countries. The authors are independent of Sun microsystems, Inc. All other marks are the property of their respective owners.

[2] A technology enabling to embed and to run, in a single threaded environment, several applications, called *applets*. Applets are programs written in a subset of Java specialized for memory constraint devices.

Data Confidentiality. Each applet in a Java Card runs in its own closed sandbox and does not access data of any other embedded applet. This is guaranteed by the firewall. To preserve this memory space separation property in the off-card storage repository, the SSMU must prevent an applet from examining or accessing data of any other applet. It should thus verify the identity of applets (*i.e.* check at runtime the client applet AID, a unique identifier on a card for each applet) to ensure that only the applet which swapped the data can recover them. For this purpose we provide each applet with a specific encryption key (called $MasterKeyAppx$ in Fig. 1). Nevertheless, since an applet can process a large number of data records and since an attacker could spy the ciphered data transmitted to the secondary storage in order to try to discover the encryption key by using cryptanalysis methods, we have chosen to cipher each data record differently. We thus provide each record with its own key (called $KeyAppxRy$ in Fig. 1) diversified from $MasterKeyAppx$.

Moreover, to ensure the Perfect Forward Secrecy (PFS) and the Perfect Backward Secrecy (PBS), our SSMU applet randomly generates a new master key each time a new applet uses the SSMU and diversifies the encryption keys for each data record to enforce confidentiality. Furthermore, a rekeying process occurs every time a record is read from the secure storage and swapped out again.

The AES algorithm in CBC mode is used to ensure confidentiality, *i.e.* the encryption and decryption processes.

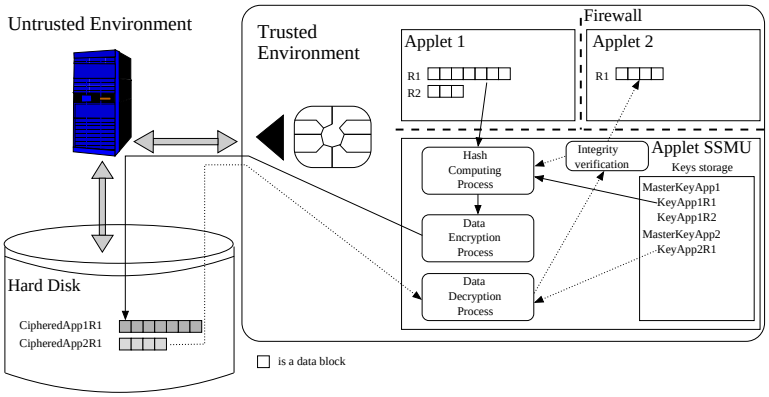

Fig. 1. Overview of the Secure Memory Extension principle.

Data Integrity. The process to ensure the confidentiality of data is simple but efficient. It consists in hashing all the input data of a record with the SHA-1 algorithm and to keep this hash value on the card. When the data is retrieved, after the decryption process, the SSMU checks its integrity by computing again the hash on the plaintext data and by comparing this value with the original one.

3 Conclusion

The Secure Extended Memory mechanism that we have designed and implemented provides an off-card swap-like feature that can enable to save more than 80% of memory, still ensuring a very good level of security by ciphering the swapped information on the flight, and by managing the keys in a way which prevents cryptanalysis.