Extended Secure Memory for a Java Card in the Context of the Java Card Grid project

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

LaBRI, Laboratoire Bordelais de Recherche en Informatique
UMR 5800 CNRS – Université Bordeaux 1
351 cours de la Libération, 33405 Talence CEDEX, FRANCE.
{serge.chaumette,achraf.karray,damien.sauveron}@labri.fr,
http://www.labri.fr/

Abstract. The Java Card[TM1] Grid[2] platform is a project carried out at LaBRI, Laboratoire Bordelais de Recherche en Informatique. It consists in setting up a platform composed of a large number of smart cards connected together by USB hubs and driven by PCs. The Java Card readers are assembled in a wall mount rack and are organized in a cluster-like architecture. The goal of the Java Card Grid project is to provide a hardware and software environment to experiment on security demanding applications and thereafter to propose innovative high level security solutions for distributed applications. This original Java Card Grid can be seen as a proof of concept (even if it is much more than that and can be used as is). Other platforms based on it are now being investigated among which:

- a network based Java Card Grid that uses cards available at various sites on the network;
- a mobile Java Card Grid [1] that uses a multilevel approach that combines ad hoc communication between mobile phones and the use of the GSM network.

The smart cards are very efficient at insuring security. Unfortunately, the main handicap of a smart card remains its limited resources. This restriction principally affects the memory size and the processor speed. Regarding memory, even if it has grown significantly in the recent years, it is still too limited to run real life complex applications. In this paper we propose a solution that makes it possible to provide a smart card with a large storage capacity, still insuring a high level of security. We show how this off-card storage solution will be useful to support real size applications on our Java Card Grid.

KEYWORDS: *Smart Cards, Java Cards, Grid, Secure Storage, Encryption/Decryption Systems.*

1 Introduction

Security is one of the main concerns in the majority of the real life applications. To experiment on these problems we have set up a grid of Java Cards [2–5]. In this context we

* LaBRI (Bordeaux, FRANCE) and University of Sfax, ENIS, TUNISIA.
** XLIM UMR 6172 CNRS – Université de Limoges (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] The Java Card Grid was awarded as the best innovative technology at e-Smart2005.

have already developed a few applications that take advantage of this testbed infrastructure. Nevertheless, one of the main limitations to the deployment of real life applications in this environment is the limited memory of the smart cards. To overcome this problem we propose a solution that makes it possible to store the application data off-card in a secure way that ensures their confidentiality and their integrity. The aim of this paper is to describe the Secure Storage Memory Unit (SSMU) application used to extend the memory capabilities of the Java Cards of our grid.

In section 2 we present the Java Card Grid and its capabilities. Section 3 then describes all the characteristics of the Secure Storage Memory Unit and shows its usefulness. Before concluding we eventually present in section 4 the applications and, specially, a data mining application already deployed on our grid that will benefit from this new extended secure memory.

2 The Java Card Grid platform

The goal of the Java Card Grid project is to provide a secure hardware and software environment to experiment and thereafter propose innovative high level security solutions for distributed applications. Our assumption is that the next generation of computers will use a hardware-supported secure environment to provide applications with a high level of security. It is a realistic assumption as can be seen when considering for instance the Trusted Computing Group [6] which develops specifications to embed a Trusted Platform Module (*i.e.* a tamper proof module similar to the smart cards of our platform) in computers. This group gathers all the leaders of the smart cards, the computer software and the computer hardware markets. There is no doubt that the technology of smart cards or at least a smart card like technology will be integrated into future computers.

2.1 Overview of the Smart Card Technology

A smart card is composed of an integrated circuit (the chip) and a plastic card of credit card size. This is a real full embedded tamper-resistant computer based on a microprocessor and different types of memory. The smart card properties, such as physical and hardware properties, are specified by the ISO7816 standard [7]. The integrated circuit includes various protection mechanisms (at software and hardware levels) such as a secure and efficient crypto-processor, sensors, a shield to avoid the electromagnetic emission and to protect against laser fault injection, a pump of current to smooth the power consumption, etc. Thus, a smart card can protect its contents and provide security services, *e.g.* ensure confidentiality of information (code and data) present in its memory. At software level, the smart card today can embed several applications on the same piece of plastic. These applications can be installed dynamically and executed independently but not in parallel. To prevent attacks from an embedded application against an other one some security features have been integrated in the operating system (*e.g.* a firewall between the applications to isolate them in a sort of sandbox still keeping the possibility to perform secure sharing of data if required). One of the multiapplication smart card technology is Java Card. Its architecture consists in a virtual machine which interprets the bytecodes of the running application (so-called applet). This technology is widely used for the Subscriber Identity Module (SIM) cards in the GSM[3]. However, the main cons of the smart card are the extremely limited processor speed

[3] The Global System for Mobile Communications (GSM) is the most popular standard for mobile phones in the world.

and the memory size. Even if it is planned that in the next two years a smart card will embed 1GB of memory (and 300 MhZ RISC processors), a solution to manage voluminous data is needed today, especially in the context of the Java Card Grid presented below. The proposed solution to overcome this limitation is presented section 3.

2.2 Hardware Description of the Java Card Grid Platform

The hardware platform that we have set up is presented figure 1.

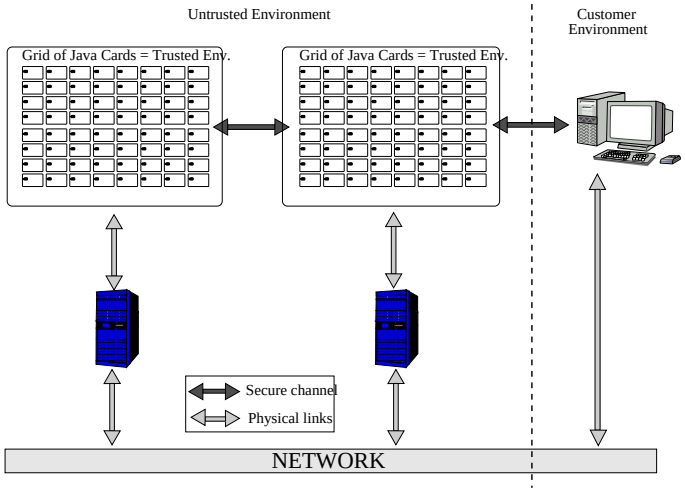

Fig. 1. A hardware platform based on Java Card grids.

It shows that we have in fact deployed two grids that are connected together by the network. The hardware fits in a wall mount cabinet of 19U. Each grid is composed of:

- a PC which needs 2U;
- two 2U racks from SmartMount, each one having 8 CCID[4] [8] readers from SCM Microsystems, i.e. we have a total of 16 CCID readers;
- three USB 7-port hubs (put in a empty 2U rack) to connect the readers to the PC and to power the readers;
- Java Cards of different manufacturers plugged in the readers which then power them.

We have also equipped one of the PCs with a LCD monitor and a special rackable keyboard (with an integrated touchpad) that we use to control the servers. A picture of this platform is shown figure 2.

Information about the software infrastructure is available in [9].

[4] CCID is a standard that defines a protocol to handle USB readers.

Fig. 2. The Java Card grid.

3 Secure Storage Memory Unit

In order to overcome the memory size restriction of the smart card, we propose a solution [10] that offers smart cards a secondary storage (the hard disk of a host) which increases the memory capacity for the embedded applications:

- in the short term, i.e. for short lived data, in a way similar to a usual virtual memory mechanisms, as can be found in most operating systems (such as Windows, Linux, etc.);
- in the long term, i.e. for long lived data, as with secure data repositories.

The proposed solution furthermore ensures the security of the swapped information, *i.e.* it preserves the integrity and the confidentiality of data to be in a perfect adequacy with the security requirements which characterize the smart card.

Two entities are involved to achieve this goal:

- one on the card, a Secure Storage Manager Unit (SSMU);
- one on the host which performs the following operations:
  - write the ciphered data sent by the SSMU,
  - manage the identification of ciphered data records,
  - return back data to the card on demand.

3.1 Principle

In this paper we focus on the entity deployed on the smart card side, the SSMU. It is useful for the applications which require to store a large amount of data. Since the smart card memory size does not enable to store such a quantity of data, the SSMU approach will enable to overcome this restriction. It is furthermore in charge of ensuring the confidentiality and the integrity of the swapped data. Using the SSMU will therefore provide a good security level even though some of the application data are stored off-card.

On a Java Card, each applet runs in its own closed sandbox and does not access data of the other embedded applets. This rule is guaranteed by the firewall provided by the Java Card Runtime Environment (JCRE) [11]. To preserve this space separation property in the off-card storage repository, the SSMU must prevent an applet from examining or accessing data of any other applet. The SSMU should verify the identity of applets to ensure that only the owner applet which swaps the data can recover them. For this purpose we provide each applet with a specific encryption key (called $MasterKeyAppx$ in Fig. 3) as described below. The problem is that an applet can process a large number of data records and 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. The best solution would be to cipher each data record differently, *i.e.* to provide each record with its key (called $KeyAppxRy$ in Fig. 3). This is that we propose below.

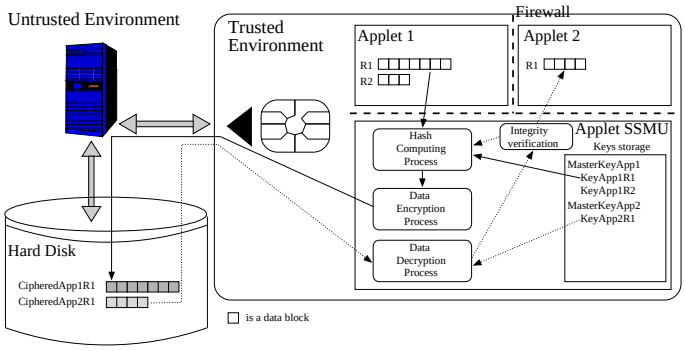

Fig. 3. Overview of the SSMU principle.

The process illustrated figure 3 is used to ensure the integrity of the swapped data and to ensure their confidentiality. These two points are respectively presented in section 3.2 and section 3.3. For short, the applet which needs more storage space, calls through the firewall a method of a shareable interface of the API provided by the SSMU applet to give it its data to save. When the source applet needs its data back it calls an other method of the API provided by the SSMU applet. The applet can also delete the swapped data. All the secure aspects are delt with by the SSMU applet.

3.2 Data Integrity Process

The process used to ensure the confidentiality of data is very simple but efficient. It consists in hashing all the input data of a record with the SHA-1 [12] and to keep this value on the card. When the data will be retrieved to be used in the future, after the decryption process the SSMU will verify its integrity, by computing again the SHA-1 on the plaintext data and by comparing this value with the original saved value. If the values match, it means that the data integrity has been preserved. Else, it means that the encrypted data was modified or corrupted for some unknown reason.

3.3 Data Encryption Process

The data encryption process that takes place in the smart cards is fundamental to ensure the privacy of the swapped information. Without this encryption, an intruder could read the data transmitted to the secondary storage. We use the AES algorithm in CBC mode.

Advanced Encryption Standard In 2000, the Rijndael algorithm was adopted as a standard by the National Institute of Standards and Technology (NIST) and named Advanced Encryption Standard (AES) [13] algorithm. It was chosen because of its low memory consumption, its significant degree of parallelism, its fast main configuration and its facility of execution. In short, the AES algorithm is a fast, efficient and strong symmetric key encryption/decryption algorithm. It is able to use 128, 192, and 256 bits cryptographic keys to encrypt and decrypt data in blocks of 128 bits (*i.e.* 16 bytes).

We use the AES algorithm as a representative algorithm. Even though the Data Encryption Standard (DES) [14] could also be applied, its use is not recommended since it is vulnerable to brute force attacks due to its small 56-bit key. Furthermore the AES algorithm is faster and offers a better performance than the DES algorithm. Triple-DES avoids the problem of a small key size, but it is unsuitable for limited-resource platforms.

Building the Encryption Key As explained above, each applet must be associated with a specific key. For this purpose we get at runtime the AID (Applet IDentifier, a unique identifier on a card for each applet) of the active applet to check the application which swaps the data. Furthermore, the SSMU associates a unique random vector of bytes, $MasterKeyAppx$, for each applet, which is used to generate the cipher key to guarantee that the data provided from different applets are ciphered in a different way. The $MasterKeyAppx$ is generated randomly using the hardware Random Number Generator (RNG) module available on the card. We have chosen to use 128 bits (*i.e.* 16 bytes) key. As explained above, each record must have a unique encryption key, but this requires a secure and large enough storage space to store these keys. The memory of the card is the perfect candidate in term of security, but not in term of size. To solve this problem we assign to each data record two additional byte values, $V1$ and $V2$, generated in a random way, to diversify the $MasterKeyAppx$ into a specific key. To generate the $KeyAppxRy$ used to encrypt/decrypt a data record Ry, we have chosen to hash with SHA a XOR of the $MasterKeyAppx$ of the applet (16 bytes) and of a combination of $V1$ and $V2$ (16 bytes) of this record Ry; we eventually keep the 16 first bytes (1).

$$KeyAppxRy = SHA_{1-16}(MasterKeyAppx \oplus \{V1V2V1V2V1V2V1V2V1V2V1V2V1V2V1V2\}) \quad (1)$$

Virtually, $KeyAppxRy$ is present on the card to encrypt/decrypt the record Ry but in reality only the $MasterKeyAppx$, $V1$ and $V2$ are stored. Thus, for each data record, two bytes $V1$ and $V2$ are stored in the card, instead of a 16 bytes key $KeyAppxRy$. Nevertheless we need to compute the key before encrypting or decrypting data.

The use of $V1$ and $V2$ values ensures that even if a key used to cipher a record is compromised, it does not affect the security of any other swapped data record *i.e.* an attacker cannot decrypt any other record by using the revealed key. We use SHA to diversify the keys because it is a good pseudo-random number generator [15].

CBC Mode The basic behaviour to implement a block cipher algorithm consists in separately encrypting each block and in gathering the partial results thereafter. This is refereed to as the CBC mode [16]. In this mode a feedback method is applied, *i.e.* the result of ciphering the previous block is used as entry to encrypt the current block. The CBC scheme is the following : the plaintext is XOR-ed with the previous ciphered block and the result is ciphered. The ciphered data of the current block will be used in the encryption of the next block.

The CBC mode solves the problem of the identical input data. Indeed, the same block is encrypted differently depending on the previous blocks. Thus, the CBC mode avoids the strong correlation between data contrary to the ECB mode where the same data is always encrypted in the same way. Moreover, in CBC mode, the order of ciphered text blocks affects the decryption phase. The valid decryption of a ciphered block requires the decryption of the previous block.

The ciphering process of the first block involves a n-bits initialization vector, denoted IV(*cf.* figure 4). To ensure a good security level, this IV must be composed of random values. In our case, to save memory, we have chosen to generate this IV from the random vector $MasterKeyAppx$ used to build the cipher key and from a combination of $V1$ and $V2$. First, we calculate the complement of $V1$ and $V2$ denoted $\neg V1$ and $\neg V2$. Second, we XOR the $MasterKeyAppx$ vector and the $\neg V1$, $\neg V2$ values. Finally, we apply the SHA hash function to obtain the IV value and we keep the 16 first bytes (2).

$$IV = SHA_{1-16}(MasterKeyAppx \oplus \{\neg V1\neg V2\neg V1\neg V2\neg V1\neg V2\neg V1\neg V2\neg V1\neg V2\neg V1\neg V2\neg V1\neg V2\}) \quad (2)$$

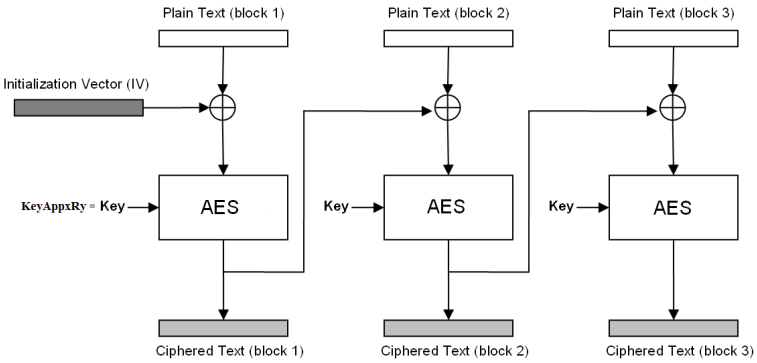

Fig. 4. The Encryption process.

Choice of Record Length We can choose any length smaller than the length of the APDU[5] array, the constraint being that we have to put the ciphered record in the data field of the

[5] The Application Protocol Data Units represent the data commands and responses exchanged by a reader and a smart card.

APDU[6]. Nevertheless the record length should be large enough to optimize the execution of the SSMU and the saved memory.

Analysis of the Cryptographic Choices To achieve its goals, our SSMU must ensure two properties: the Perfect Forward Secrecy (PFS) and the Perfect Backward Secrecy (PBS). The PFS guarantees that a passive adversary who knows a contiguous subset of keys cannot discover subsequent keys. The PBS guarantees that a passive adversary who knows a contiguous subset of keys cannot discover preceding keys. The first mechanism that we have used to achieve this goal is the random key generation which occurs when a new applet uses the SSMU. The additional random generation of V1 and V2 for each record enforces the confidentiality since these values and the applied operations diversify the encryption key at record level (which is a subset of the applet level). Indeed if the attacker succeeds in computing the encryption key for a given data record, this does not affect the confidentiality of the other data records. Furthermore a rekeying process occurs every time a record is read from the secure storage and swapped out again.

3.4 Total Separation Between the Data of Different Applets

Java Card is a multiapplication card, *i.e.* it can contain several applications at the same time and applets can be downloaded and removed dynamically during the life cycle of the card. For security reasons, each applet in a card must run in its own closed sandbox, so that it cannot interfere with the execution of any other applet and cannot access confidential data. To achieve this goal, the Java Card approach is to use a firewall that guarantees a total separation between applications and thus offers a protection from malicious codes.

To be consistent with the above principles, the SSMU must insure that an applet cannot access (an unciphered version of) the swapped data of any other applet. Therefore, when receiving a request from an applet that wants to get a piece of data back in the card, the SSMU verifies the identity of the requesting applet. So as to make this verification possible, the SSMU saves the AIDs[7] (Applet IDentifier) of applets when they swap data out of the cards. Thus, an applet can only recover its own data, and it has no way to access any swapped data of any other applet.

Unfortunately, if an applet does not delete its swap data before it is removed, a new applet installed with the same AID can easily access and examine them in the swap area of the old applet. There are two ways to prevent that problem. The applet can delete its swapped data by calling a dedicated method of the API provided by the SSMU unit. The applet can also implement the *AppletEvent* Interface [8] provided by the Java Card API. The *uninstall()* method contained in this interface is called when the applet is deleted and makes it possible to inform the SSMU so that it can remove the swapped data.

[6] Even though in practice the size of the APDU buffer depends on the card brand, it is usually 256 bytes. This is large enough for the applications presented in section 4.

[7] The AID is the unique identifier of an applet on a card.

[8] This interface is available only in Java Card version 2.2 and later.

3.5 Memory Saved by our Swapping Strategy

Using the SSMU enables to save a lot of card memory space. Inside the card, a swapped record is represented by a Java object containing all the pieces of information (attributes) and methods needed to decipher the record when the ciphered data is returned back to the card (*cf.* Listing 1.1).

Listing 1.1. A data record.

```
public class record {

    short lengthPlainData =(short)0;
    short lengthCiphereData = (short)0;
    short index = (short)(-1);
    boolean empty = true;
    short externalAdress =(short)0;
    byte[] hash = new byte[20];
    byte v1;
    byte v2;

    public void newRecord(short lengthData, ...)
    public boolean isEmpty()
    public void delete()

}
```

If we note x the length of the swapped data record, the size of the data needed to represent it on the card (*i.e.* the object and the MasterKeyAppx which is required to decrypt the record) is: $31 + 16/n$ where 31 is the length of all the fields of the object (a short is 2 bytes long) and 16 is the length of the $MasterKeyAppx$ and n the number of records associated to the applet (the $MasterKeyAppx$ is shared). Thus the rate of saved memory is: $\frac{x-(31+16/n)}{x}$. For example, for one application we can save:

% of saved memory	n **record**	x **length of each record**
0	1	47
87,20	16	250
100	n	$x \rightarrow \infty$

Note that 0% means “no memory saved”. Thus, as soon as the record size is greater than 47 bytes, we save memory by using our swapping mechanism (first line of the above table).

4 Applications

In this section we present three applications for our SSMU. The first and obvious consists in integrating this feature in one of our existing applications. The second is a scenario that could be used in different user cases and for instance in a real grid environment. The third is a new feature for our Java Card Grid. It should be noted that the current interface between a card and the outside world is quite slow. The consequence is that the applications described

here will not run fast on large size data. Nevertheless, we believe that future cards will provide a faster interface.

4.1 Using the SSMU for secure data-mining in the Java Card Grid: the CBP-Air France application

This example is related to an application that we have developed on the Java Card Grid platform. Even though it does not currently make use of the SSMU feature, it is clearly the kind of application that can take advantage of it. We are working on the implementation of this topic.

The context of the application is as follows. For security reasons, the Transportation Security Administration (TSA) of the United-States is developing a passengers analysis program called Secure Flight. This program is fed with the passengers information that has to be provided by airline companies for any flight for which the origin or destination port is in the U.S. In some cases, this raises problems, for instance with the European laws that forbid these companies to reveal (pieces of) passenger information, in order to protect privacy and civil liberties.

This may lead the American authorities to refuse the landing of such flights if they cannot access the passengers files. To solve this challenge, we have developed an application that runs on and uses the features of the Java Card Grid platform and data mining techniques to examine the passengers file still preserving the privacy of the concerned passengers. This application makes it possible to analyze the information contained in the files of an airline company, for example Air France, by a government agency, for example the office of the Customs and Border Protection of the United States, the CBP. Our goal is to ensure the privacy of information for both organizations. For Air France, we must guarantee the confidentiality of the sensitive data present in the Passenger Name Records (PNR). For the CBP, we must ensure that the algorithm and the search criteria cannot be revealed. The CBP must thus be able to search for suspect passengers in the file of Air France without being able to identify any specific passenger. Air France must not be able to discover the criteria used by the CBP.

To deal with these constraints, Air France deploys its PNR over the cards of the Java Card Grid and provides an interface (*i.e.* API) giving partial or full access to the passengers information without violating the confidentiality requirements. In particular, a passenger is identified by a `key` that is independent of his effective identity. The code of the CBP is also deployed on the cards; it carries out its search and returns back the `key`s of the suspect passengers. The CBP can then ask Air France the identities of the suspected passengers. More details about this application are available in [4, 5, 17–19].

In this example, the SSMU applet will be useful because the PNR of Air France can be bigger than the overall memory size of all the cards of the grid. Thanks to this mechanism, we will be able to store the passengers file in a big capacity storage, still respecting the confidentiality and integrity requirements of the swapped data.

4.2 A Secure Storage Application

The goal of this application is to provide an efficient and fast secure storage solution for the end user. An illustration of this application is a user of a PC who wants to securely store his data. We assume a client application, running on the host, that wants to manage a virtual secure file system. To achieve this goal it uses a dedicated service provided by our grid.

This service consists in ciphering the data sent by the client application of the PC using the SSMUs of the grid in a transparent way. Information about the services possibilities in our grid is available in [20]. This kind of application can be useful for many user cases where it is needed to have a secure storage.

Now we can consider to use it to perform the secure storage tasks of a real grid. This application can then be viewed as a sort of collaborative process between two grids, the real one and the Java Card Grid, each performing what they are better at, the scientific computations for the real grid, and the ciphering process and the keys protection for the Java Card Grid. Note that smart cards have extremely efficient encryption dedicated hardware and that the encryption key(s) can remain inside the card and never be communicated to any outside entity. The development of this application is not yet started but it could be quickly achieved.

4.3 A Secure Card Applications Repository

A useful feature for our grid would be a secure repository of card applications, that would consist in storing the CAP[9] files in a secure place to ensure their confidentiality and integrity. When a card will need a new application that it does not yet embed to provide a service, it will get the related CAP file in this secure repository and it will install it in a secure way. Moreover this special operation of getting a service and installing it can be seen in our grid as a dedicated distributed service offered by some cards to other cards. Currently this application is not yet available because we need to implement a card embedded GlobalPlatform [21] installer for the server card (*i.e.* the card which will decipher and send the CAP file to the client card – *i.e.* the card which will ask for the new application) and this is a hard and long task.

5 Future work : security analysis of the solution

So as to analyse our solution in terms of security[10] we will initiate a joint work with cryptologists. We will especially study:

- the relationship between keys in our solution;
- the relationship between the IV and the encryption keys;
- the choice of cryptographic primitives,
- the influence of the recurrent characteristic of the keys and IV.

Furthermore, one of the referees for this paper wrote: *"It is maybe worth noting that given a high assurance scenario (e.g. eID card or possibly the new travel regulations) all security features required by the proposed application(s) would be evaluated for security to resist high attack potential.*", which we agree with. This would furthermore increase the security assurance of our proposal.

6 Conclusion

The framework of this paper is the Java Card Grid project developed at LaBRI, University Bordeaux 1. It consists in providing a platform based on a number of connected Java Cards

[9] The standard binary file format for the application of the Java Card platform.

[10] We need to thank the referees that suggested to investigate further in this direction

to experiment on secure computing. One of the major limitations of this platform is the small memory size of the Java Cards. It is thus a problem to develop distributed applications that have to cope with a large amount of data.

In this paper we have presented what we call the Secure Storage Memory Unit, a mechanism that provides a swap like feature for Java Cards. This mechanism makes it possible to swap data off-card still ensuring a very good level of security by ciphering the swapped information. The swapped data is split into records, each record having its own symmetric ciphering key that is furthermore built on the flight. Managing keys this way has two advantages: it prevents cryptanalysis; it saves space in term of card memory by building on the flight the required keys. We have shown the amount of memory that can be saved by using our SSMU. We have also described applications that could benefit from this feature and now, implementing them is the next step of our work.

Thanks

Our project is supported by:

- Axalto, Gemplus and IBM BlueZ Secure Systems (for the cards);
- SCM Microsystems and SmartMount (for the readers);
- Sun microsystems (for the overall platform).

We also thank: Fujitsu, Giesecke&Devrient, Oberthur Card Systems and Sharp for the Java Card samples; David Corcoran and Ludovic Rousseau for their work on `pcsc-lite` and the CCID generic driver.

References

1. Chaumette, S., Markantonakis, K., Mayes, K., Sauveron, D.: The Mobile Java Card Grid Project. In: Proceedings of e-Smart 2006, Nice, France (2006)
2. Chaumette, S., Grange, P., Sauveron, D., Vignéras, P.: Computing with Java Cards. In: Proceedings of CCCT'03 and 9th ISAS'03, Orlando, FL, USA (2003)
3. Chaumette, S., Sauveron, D.: The Smart Cards Grid Project. `http://www.labri.fr/Perso/~chaumett/recherche/cartesapuce/smartcardsgrid/documents/poster.pdf` (2003) Poster presented at Cartes 2003.
4. Chaumette, S., Grange, P., , Karray, A., Sauveron, D., Vignéras, P.: Secure distributed computing on a Java Card grid. In: Proceedings of 7th International Workshop on Java for Parallel and Distributed Computing, Denver, CO, USA (2005)
5. Atallah, E., Chaumette, S., Darrigade, F., Karray, A., Sauveron, D.: A Grid of Java Cards to Deal with Security Demanding Application Domains. In: Proceedings of e-Smart 2005, Nice, France (2005)
6. Trusted Computing Group: Trusted Computing Group Home. `https://www.trustedcomputinggroup.org/home` (2006)
7. International Organization for Standardization: ISO7816. `http://www.iso.ch/` (2005)
8. USB Implementers Forum: Universal Serial Bus Device Class Specification for USB Chip/Smart Card Interface Devices version 1.00. `http://www.usb.org/` (2001)
9. Chaumette, S., Karray, A., Sauveron, D.: The Software Infrastructure of a Security Platform Java Card based for the Distributed Applications. In: Submitted. (2006)
10. Chaumette, S., Karray, A., Sauveron, D.: Secure Extended Memory for Java Cards. `http://damien.sauveron.free.fr/publications/ICCSA2006CRPoster.pdf` (2006) Poster presented at the 2006 International Conference on Computational Science and its Applications (ICCSA 2006).

11. Sun microsystems: Java Card™ 2.2.1 Runtime Environment (JCRE) Specification. Sun microsystems (2003)
12. Science, Technology: FIPS 180-2, Secure Hash Standard (SHS). Technical report, Federal Information Processing Standards Publication 197 (2002)
13. Science, Technology: Advanced Encryption Standard (AES). Technical report, Federal Information Processing Standards Publication 197 (2001)
14. Science, Technology: FIPS 46-3, Data Encryption Standard (DES). Technical report, Federal Information Processing Standards Publication 197 (1999)
15. Filiol, E.: A new statistical testing for symmetric ciphers and hash functions. Cryptology ePrint Archive, Report 2002/099 (2002) http://eprint.iacr.org/.
16. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: 7. In: Handbook of Applied Cryptography. CRC Presss (1996)
17. Chaumette, S., Grange, P., , Karray, A., Sauveron, D.: Gestion de la Sécurité pour l'Extraction Parallèle Distribuée des Connaissances. In: Proceedings of EGC 2006 (Sixième Journées Francophones "Extraction et Gestion de Connaissances"), Lille, France (2006)
18. Sauveron, D.: Étude et réalisation d'un environnement d'expérimentation et de modélisation pour la technologie Java Card. Application à la sécurité. PhD thesis, Université Bordeaux 1 (2004)
19. Karray, A.: Calcul sécurisé sur grille de cartes à puce. Master's thesis, ENIS – University of Sfax (2004)
20. Chaumette, S., Karray, A., Sauveron, D.: Security of Collaborative and Distributed Services in the Java Card Grid Platform. (In: Proceedings of the Workshop on Collaboration and Security (COLSEC'06))
21. GlobalPlatform: GlobalPlatform. (http://www.globalplatform.org/)