Which Security Policy for Multiapplication Smart Cards?

[Published in *Proc. of USENIX Workshop on Smartcard Technology*, pp. 21–28, Usenix Association, 1999.]

Pierre Girard

GEMPLUS, Cryptography and Security R&D
Parc d'Activités de Gémenos, B.P. 100, 13881 Gémenos CEDEX, France
pierre.girard@gemplus.com

Abstract. In this paper, we aim to clarify some issues regarding the deployment context of multiapplicative smart cards. We especially deal with the trust relationships between the involved parties and the resulting constraints from a security point of view.
We highlight a new security threat in a multiapplicative context and propose a new multilevel security model which allows to control precisely the information flows inside the card, and to detect illegal data sharing. Finally we illustrate all the proposed concepts on an multiapplicative example involving three applications.

1 Introduction

Multiapplication smart cards are getting more and more attractive for numerous good reasons. Users are willing to reduce the number of cards in their wallets, issuers want to decrease the time-to-market, the development, infrastructure and deployment costs or to update/add applications after card issuance. In addition multiapplication smart cards allow commercial synergies between partners and can lead to new business opportunities. A credit card with an electronic purse and a frequent flyer application is a classical example of a multiapplication smart card.

A few operating systems have been proposed to manage multiapplicative smart cards, namely Java Card,[1] Multos[2] and more recently Smart Card for Windows.[3] In this paper we will focus on Java Card and exhibit examples for this platform, but all the results can apply to any multiapplicative platform.

Security is always a big concern for smart cards, but the issue is getting more intense with multiapplicative platforms and post issuance code downloading. Of course, Java Card security or protection against aggressive applets have

[1] See http://java.sun.com/products/javacard.
[2] See http://www.multos.com.
[3] See http://www.microsoft.com/smartcard.

been discussed extensively, but we still lack a global security policy for multiapplicative smart cards. We will show that this kind of policy must be more than the simple concatenation of the security policy of all applications. Until now, this topic hasn't been appropriately investigated, probably because numerous issues concerning how those cards will be used remain unclear: which parties will cooperate? how? which of them will drive the scheme? etc.

In this paper we first aim at clarifying those issues. Next, we show that there is a need for a card-wide security policy, mainly because of the existance of information sharing mechanisms between applets. Then, we propose a new security policy to deal with this need and show an example of how it could be used. We finish with limitations, potential extensions and related work.

2 The Multiapplication Platform

On a multiapplication platform, we usually find an operating system managing the card resources (like I/O, memory, random number generator, crypto engine...) and some applications (possibly loaded after the card issuance) from various sources using the OS services. In addition, the OS ensures application segregation and provides mechanisms to allow controlled data sharing between applications.

2.1 Which participants?

Unlike single-application smart cards, multiapplication smart cards involve many participants. Of course, there are still an issuer and an end user, but additional third parties which can interact directly or not with the card are also involved.

To separate precisely the actions of the participants we have defined two roles in addition to the usual issuer which is played by a unique authority: the application provider and the card operator.

The application provider designs an application for the targeted card operating system and negotiates with the issuer for downloading its application inside the card (before or after the card issuance). It's the owner of the applet and applet's data. Of course, the issuer itself will place some applications inside its card, and will play the application provider's role.

The card operator is an entity which can interact with the card either to use an application or to perform administrative tasks. An administrative task can be anything from auditing the card or updating a key to downloading a new application. The interaction between the operator and the card can be direct (e.g. if the card is inserted in an ATM) or remote (e.g. through the Internet or a cellular phone network). It is likely that the application providers and the issuer will play the operator's role as well. Conversely, an operator will probably be an application provider, even if it could be delegated to interact with an application of another provider. An applet can be designed to work with one operator (likely its owner) or more. In this case its behaviour could differ according to the operator.

Coming back to the example of a credit card with a frequent flyer application, we can assume that the issuer will be a bank, which will also be the application provider and the operator of the credit applet. An air travel company will be the provider and the operator of the loyalty applet. Some other companies can join the loyalty program and become operators for the loyalty applet.

2.2 How is it operated?

The first and the simplest way to operate such a multiapplication platform is to keep it entirely under the control of the issuer. It will be the only authority responsible for the card and its integrity. All application providers must negotiate and agree with the issuer conditions and guidelines for downloading their applications on the card. The issuer is granted all possible rights in the card and will enforce its policy during all the card life, as well as inspect applications carefully before downloading them. In the rest of the paper we suppose this mode of operation as it seems to be the most likely in the forthcoming years. However, other ones like the two following could exist.

One can imagine a second model where the end user buys a blank card from a manufacturer of its choice and plays the issuer's role. Afterwards, it will accept or buy from providers some applications for its card. The whole card will be (or should be) under its control. We do not consider this scenario as it is unclear if providers will accept to download their applications into cards for which they have no or few behaviour guarantees. Probably, this type of scheme will coexist with the first one, but for non security critical applications or for applications totally operated by the end user. For example, a manufacturer of smart locks using smart cards, instead of keys, could propose to download its application on users cards. The users could decide to buy an *ad hoc* card or use an existing one to put the application driving the smart lock.

A third type of scheme adds another role: certification authorities. Those authorities will audit issuers and their cards and will provide a certification which will guarantee that an issuer respects a given policy. Based on this policy a provider as well as an end user will be able to decide if the issuer's policy complies with their requirements. An example of certification could be a "privacy awareness" label for issuers which respect the privacy rights of the users and do not leak private data outside the card. This is a refinement of our first scheme and we won't focus on it in the following.

2.3 Which security policy?

At first glance, it is easy to see that there are, at least, two separate security problems: the platform level security and the application level security.

The first one (platform level security) concerns applications segregation (this can be viewed as the classic OS security) as well as the quality of security services offered by the platform (e.g. correctness of the Java virtual machine including the verifier, tamper resistance, cryptographic algorithms and post-issuance loading mechanism). This part is under the issuer's responsibility.

The second one (application security), is under the provider's responsibility, but relies necessarily on the platform security. Moreover, the application should assume that the OS won't be aggressive and will act as it is supposed to. Conversely, the OS doesn't make any assumption about the application and should still work and protect the other applications even if an aggressive or unsecure piece of code is loaded. However, one should note that even if this is technically perfectly acceptable, and if an unsecure application can't threaten the platform or other application, the end users or potential customers could get confused by a break-in of an application and mixed up the platform security and the application security. To avoid this potential damage to its brand image, an issuer could enforce a minimum security level for the applications loaded on its card by reviewing them or operating a scheme including some certification authorities.

Apart from these two obvious security aspects, a third one must be addressed. All the difficulties arise from data sharing inside a card. Actually, most of multiapplication smart cards, in order to build cooperative schemes and optimize memory usage, allow data and service sharing (i.e. objects sharing) between applications. And beyond this point there is a need for a card-wide security policy concerning all the applications. A small example should make this point clearer. When an application provider A decides to share (or more probably to sell) some data with an application provider B, he will ask for guarantees that B won't be able to resell those data or make them available world-wide.

To address these problems two kinds of security policies can be introduced: a discretionary one and a mandatory one. The applications will be in charge of defining their own discretionary security policy which could be enforced by the OS. For an example, in a Java card, an applet can decide to share some of its objects with a selective list of other applets. On this access control list basis, the OS will allow or deny access to the shared object by other applets. If we just consider a discretionary policy, nothing prevents an application B which could legally access an object of A (B has been granted from A to read the data) from copying the information into another object shared with C. In this case the information is transfered from A to C even if C is not granted access the information from A.

A mandatory security policy is necessary to solve the problem of re-sharing shared objects as pointed above. Actually none of the existing OS enforce such a policy.

In this paper, we will discuss this last point and propose a security model, compliant with the requirements of safe data sharing and able to control the information flows inside the card.

2.4 Which threat model?

In the following we consider the threat of an insecure or malicious applet gaining legally some access to sensitive data (by a sharing mechanism of the OS) but resharing them illegally with a third party.

The threat could be a commercial concern (as in the previous example) or privacy concern. This later case is especially important when dealing with health-

care cards containing medical records or with loyalty cards containing marketing profiles.

3 A New Security Model

The security policy enforced by the OS should model the information flows between the applications which, themselves, reflect the trust relationships between the participants of the applicative scheme. In this section we will start by studying the trust relationships, then the security model, and finally consider some implementation issues.

3.1 Trust relationships

In the basic situation the only trust relationships are from everyone to the issuer, as there is no way for a participant to distrust the issuer. The platform OS is completely under the issuer's control and is potentially able to read, write, create or modify everything on it including the applications and their keys.

In addition, some participants can trust other ones: sometimes because it is in fact the same entity which plays more than one role and sometimes because there is an agreement or a contract between the two.

It should be noticed that the trust relationship is neither symmetric nor transitive: an entity wouldn't like to trust someone only because one of its partners trusts it. This situation is not likely to change as cooperation in industry becomes something more complex and subject to daily change (one could speak about coopetition between companies as well as between divisions inside a company).

Figure 1 shows an example of trust relationships with four applications providers and three operators. The issuer trusts no-one except OP1 and AP1 which are two roles played by the issuer itself. We can see that AP1 trusts AP2 and AP2 trusts AP3 which doesn't mean that AP1 trusts AP3.

We can also note that AP4 doesn't operate its applet itself but relies on, and thus trusts, OP3 to do so.

Now, it should be clear that a mandadory security policy must be enforced in a multiapplicative scheme which will allow or deny data flows between applications given the trust relationships. If an entity A trusts an entity B, this means that some information could flows from A to B. We have chosen a classic multilevel security policy using a set of security level modelling the trust relationships.

3.2 The multilevel policy

The multilevel security policy, first modelled in [2], uses a set of security levels with a complete lattice structure. A security level is associated to each subject (an entity manipulating information) and each object (a piece of information but not an object in object-oriented programming sense) of the system. The

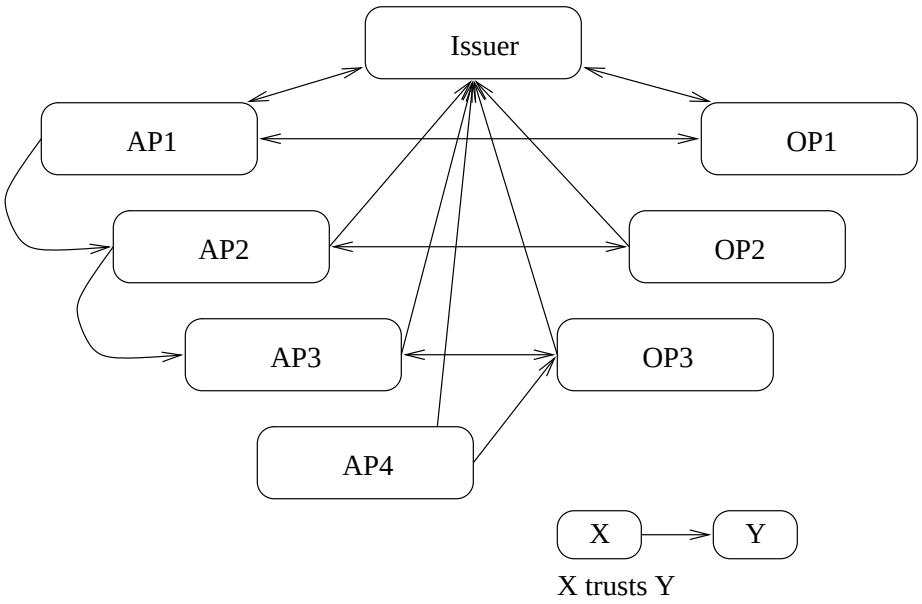

Fig. 1. Example of a trust relationship

lattice structure implies an order relation on the set, and hence that the relation is transitive.

The security rule states that a read (resp. write) access by a subject to an object is granted if and only if the level of the subject is greater (resp. lower) or equal to the object's level. The classical example of a multilevel security policy is the document classification inside a military organisation. In this case the set of security level is: {*Unclassified, Confidential, Secret, Top secret*} with a usual order between the security levels.

We will now consider how to map each entity and information in a multiapplicative scheme to a security level. First of all we will create one level associated to the issuer and one associated to each application provider. If an operator and an application provider trust each other they will be represented by only one level.

To establish a multilevel policy for a card without data sharing, one can choose a (non flat) lattice of security levels with one level for each application provider and an additional public level to complete the lattice. Figure 2 indicates the security level lattice corresponding to the example of figure 1, part (a), where an arrow represents an order relation between two levels. In this case, the only legal information flow is from a public level to the issuer through any role, but communication between providers or operators is strictly prohibited except from AP4 to AP3 because AP4 trusts OP3 and because OP3 and AP3 have been merged.

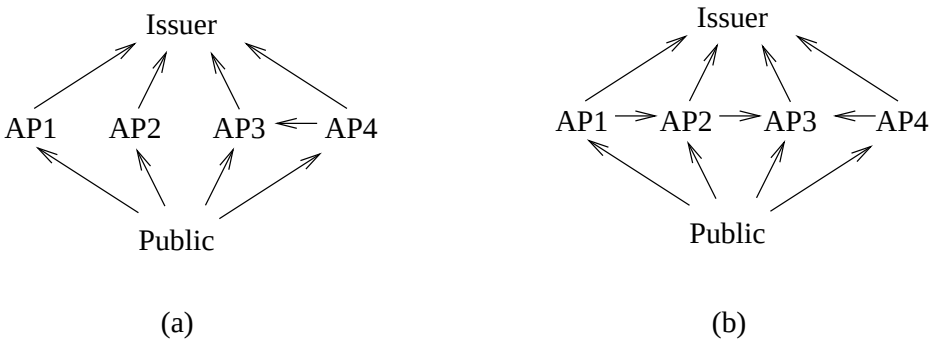

Fig. 2. Example of security level lattice without sharing (a) and with an unacceptable sharing (b).

To allow data sharing between entities, one cannot simply allow a new order relation between two roles as we cannot accept the transitivity effect of the order relation. As an example, if AP1 from our last example wants to share some data with AP2 and AP2 wants to share data with AP3, the security lattice of figure 2, part (b) is clearly inadequate. We recall that the trust relationship is not transitive and so AP1 does not want to share data with AP3 which is possible with this solution through AP2.

To solve this problem we propose that AP1 and AP2 agree on a data subset they want to share. Those data will be classified with a new level called AP1+AP2 placed in the security level lattice shown on figure 3. The same agreement will take place between AP2 and AP3 resulting in another level AP2+AP3.

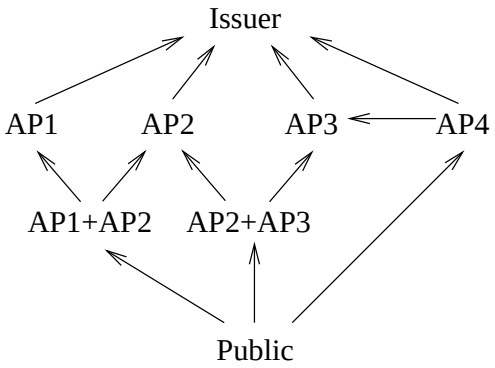

Fig. 3. Example of security level lattice with acceptable sharing

This technique can be refined if there is a need for sharing a subset of AP1+AP2 with AP3. We will create a new level called AP1+AP2+AP3 greater than the public level but lower than AP1+AP2 and AP2+AP3.

3.3 Implementations issues

To enforce the latter policy within a card (we consider here a Java Card), a lot of implementation issues should be considered. First of all, we should decide which data will be classified, and with which granularity. We should also consider the implementation of security mechanisms which will enforce the policy.

The classification of objects in object oriented language is a complex problem (see [5] for a recent discussion on this subject), however, we can chose a simple strategy by assigning a security level to each object instance. In a given applet, the objects will be labelled with their provider's level except if an object is shared, in which case, we will choose the level related to shared data. The authorized information flows in an applet will be from lower labelled objects to higher ones.

Enforcing the security policy could be done dynamically by a reference monitor (part of the card OS) which will be called each time an object reference is used by the virtual machine or statically by checking the correctness of the information flows in an applet. The first solution would be too costly in memory and execution time which are both critical in a smart card as we shoud tag each object with its level and check the validity of each read/write operation.

The second solution has been studied for a long time (Denning pioneered this domain [6] and was followed by [8, 1, 11, 4]) and could be integrated to existing static verifier of Java bytecode. Using security level set with a lattice structure is a key point to guarantee that the static analysis will terminate.

Practically, this means that an applet provider will deliver its code to the issuer along with the security level of all the objects contained in it. The issuer will verify that the code and the declared levels of the objects comply with the other applets and their objects security levels.

4 A Real World Example

We present in this section the example of a multiapplicative healthcare card. This card is issued by a health insurance with an applet. This applet contains some administrative data including the social security number of the card holder.

In addition, the user has joined the loyalty program of a drugstores chain and downloaded the corresponding applet. The drugstore applet can use the social security number and contains some administrative data and, possibly, a few medical records (e.g. medication allergy). It also contains a loyalty part which maintains a loyalty points counter.

The third applet on the user's card is a loyalty applet of a sport centers chain which has an agreement with the drugstores chain and shares the loyalty point counter with the drugstore applet.

Figure 4 summarizes the information flow between the applets.

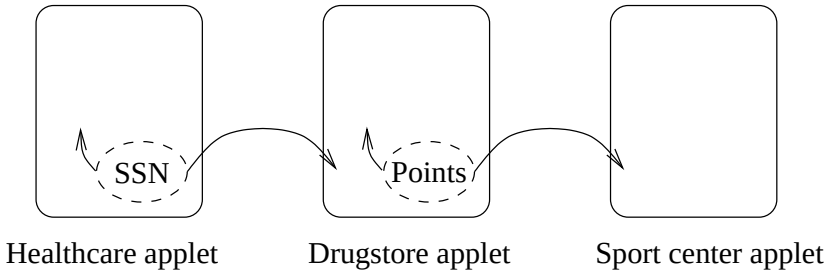

Fig. 4. Example of three applets sharing data

The drugstore applet can read the social security number, but is not allowed to give it to other applets. The security threat is the following: the drugstore applet can copy the social security number from the healthcare applet to the loyalty point counter and broadcast it to applets allowed to read the counter.

To face this threat, we propose a mandatory security policy using the security level lattice of figure 5. All the objects of the healthcare applet will be labelled with HC except the social security number labelled with SSN. The objects of the drugstore applet are labelled with the DS level except the loyalty point counter which is labelled with PTS. Finally, all the objects of the sport centers chain applet are labelled with SC.

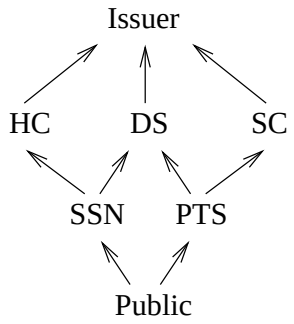

Fig. 5. Lattice of security levels for figure 4 applications

This way, if the drugstore applet copies the social security number in one of its objects (labelled DS) and later in the loyalty point counter (labelled PTS), an illegal information flow will be detected as PTS is lower than DS and information can only flow from a lower level to a greater one.

To be more complete, we should deal with the external world: the drugstore applet can exchange some data with an operator. The external software used by

the operator should also be checked to verify that it doesn't content a covert channel transfering information from the DS level to the PTS level.

As this software can be modified, the issuer can also ask the drugstore applet provider to encrypt the social security number before sending it outside the card.

5 Limitations and Related Work

The limitations of our work clearly come from the communications with the external world. We are currently improving this point of the security model. We are also in the process of implementing static verifiers of applets as well as including declassification processes (e.g. when the data flow through a ciphering filter) in the security model.

Some authors have already dealt with non-transitivity constraints in different contexts [10, 3], but we are not aware of a multilevel security policy applied to a smart card and its environment. A lot of papers dealing with classic Java Card security are available. We refer the reader to recent publications like [9] or [7] for a complete bibliography.

6 Conclusion

In this paper, we clarify some issues around the operating scheme of multiapplicative smart cards and highlight some new security threats.

The proposed multilevel security model allows us to control precisely the information flows inside the card, and detect illegal data sharings.

In a next step, analysing tools should be developed and provided to issuers which will be able to audit applets proposed by third parties for their cards.

References

1. Jean-Pierre Banâtre, Ciarán Bryce, and Daniel Le Métayer. Compile-time detection of information flow in sequential programs. In Dieter Gollmann, editor, *Proceedings of ESORICS*, number 875 in LNCS, pages 55–73. Springer, November 1994.
2. D. Bell and L. Lapadula. Secure computer systems : Unified exposition and MULTICS interpretation. Technical report ESD-TR-75-306, MITRE Corporation, 1975.
3. Pierre Bieber. Formal techniques for an ITSEC-E4 secure gateway. In *Proceedings of the Twelfth Annual Computer Security Applications Conference*, December 1996.
4. J. Cazin, P. Girard, C. O'Halloran, and C. Sennett. Formal validation of software for secure systems. In *Formal Methods, Modelling and Simulation for System Engineering*, Saint-Quentin-en-Yvelines, February 1995.
5. Frédéric Cuppens and Alban Gabillon. Rules for designing multilevel object-oriented databases. In Jean-Jacques Quisquater and *al.*, editors, *Proceedings of ESORICS*, number 1485 in LNCS, pages 159–174. Springer, September 1998.
6. Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Communication of the ACM*, 20(7):504–512, July 1977.

7. Jean-Louis Lanet and Antoine Requet. Formal proof of smart card applets correctness. In Jean-Jacques Quisquater and *al.*, editors, *PreProceedings of CARDIS*, Louvain-la-Neuve, September 1998.
8. Maasaki Mizuno and David Schmidt. A security flow control algorithm and its denotational semantics correctness proof. *Formal Aspects of Computing*, 4:727–754, 1992.
9. Joachim Posegga and Harald Vogt. Byte code verification for java smart cards based on model checking. In Jean-Jacques Quisquater and *al.*, editors, *Proceedings of ESORICS*, number 1485 in LNCS, pages 175–190. Springer, September 1998.
10. John Rushby. Noninterference, transitivity, and channel-control security policies. Technical Report CSL-92-02, SRI, December 1992.
11. Dennis Volpano and Cynthia Irvine. Secure flow typing. *Computers and Security*, 16(2):137–144, 1997.