Multi-Application Smart Cards: Card Operating Systems and Application Security
Kenneth R. Wilcox, Jr.

ABSTRACT
Multi-Application Smart Cards are becoming more prevalent in society as the need to secure electronic transactions increases for both e-commerce and identity verification purposes. This paper studies two Multi-Application Smart Cards and their Operating Systems, namely the MULTOS Card and the Sun Java Card. This research provides an in-depth analysis of the efficacy of the mechanisms used to verify applications that are installed after cards are issued. In addition to that, the application firewall mechanisms of each card are examined to determine how effective they are at securing the card applications. Finally we assess which Smart Card Operating System implements a more secure environment to host multiple applications and best supports dynamic loading and deletion of card applications.

Keywords
MULTOS, Java Card, Multi-application, security.

1. INTRODUCTION
There are two major competitors for operating systems in the smart card market today. They are MULTOS and Sun’s Java Card. MULTOS is an open, non-proprietary multi-application smart card operating system [17]. To help ensure the openness and encourage the adoption of the MULTOS operating system, the control of the MULTOS specification is contractually invested in The MULTOS Consortium [17]. The MULTOS Consortium is also known as MAOSCO, which is composed of members whose goal is to promote MULTOS as an industry standard across the market. MULTOS is a widely used card operating system for MasterCard International.

The Sun Java Card is the preferred smart card platform for Visa. Visa and MasterCard are members of the Global Platform which establishes and maintains standards for smart card interoperability. Some of the objectives of the Global Platform are backwards compatibility, application portability, reduced time to market and lower cost of smart card implementation. Visa has recently approved MULTOS to be used on their cards, although they are not a member of MAOSCO.

In this competitive market there are certain attributes of smart cards that enhance their ability to provide a safe and secure execution environment for card applications. In order to ensure the runtime environment is secure, we must examine some of the specific qualities of smart cards that have a direct impact on application security.

Application code verification is an important aspect of smart card security. It raises some questions about the code delivery mechanism for getting the code to the small credit card-like devices. The installation process of the code onto these devices must be secure, as well as the environment in which the code executes. The card issuer must be assured that if applications are installed either at issuance or post issuance there is a secure delivery mechanism that is standard. In addition, we must ensure that applications that are installed on the cards do not violate any runtime restrictions the virtual machine has in place for security reasons.

The availability of encryption algorithms also plays a significant role in securing smart card applications. The Digital Signature Algorithm (DSA) is used to ensure that data is intact and is from a valid source. Also the support of various encryption algorithms, both symmetric and asymmetric, enables application providers to choose the algorithm that is most appropriate for their individual applications depending upon the nature of the data being protected. For more sensitive data, public key technology is appropriate, where the private key is kept confidential on the card and only the public key is exposed to other parties.

Present day smart cards can run multiple applications, hence the phrase multi-application smart cards. In addition to being able to run multiple applications, smart cards can have applications installed, updated, and deleted post issuance at the request of the customer or the card issuer. It is imperative that the process to initiate card updates is secure and controlled to protect the confidentiality of the data on the card and the private data of the card holder.

1.1 Purpose
The purpose of this study is to identify if there is a significant difference in smart card application security between the MULTOS Card and the Sun Java Card. The results will provide insight to smart card OS vendors toward improving the application security mechanisms built into the card, as well as point out any weaknesses that should be addressed in future revisions of card protection profiles.

1.2 Methodology
Null Hypothesis:
H₀: There is no statistically significant difference in application security when comparing the Sun Java Card and the MULTOS card.

The hypothesis will be assessed based on the following criteria:
- Encryption mechanism (symmetric, asymmetric, key strength)
- Bytecode Verification (installation and runtime verifications)
- Common Criteria/Information Technology Security Evaluation and Certification (ITSEC)
- Post Issuance Application Installation/Deletion Processes
Each candidate is ranked on each of the above criteria using ordinal values described as follows:

- Inferior = -1
- Adequate = 0
- Superior = 1

A value of -1 signifies that one or more characteristics of the criterion are not met or not supported. A value of 0 states that the criterion is supported adequately, but there is room for improvement to meet higher standards with respect to security. A value of 1 states that the criterion is satisfied with the highest level of robustness and security.

1.2.1 Encryption
The smart cards are evaluated on the number and type of cryptographic algorithms that are supported. An inferior rating is given if the card only supports symmetric key cryptography (DES). A value of 0 is given if the card supports asymmetric cryptographic algorithms (RSA). A value of 1 is given if the card supports both symmetric and asymmetric key cryptography as well as digital signatures.

1.2.2 Bytecode Verification
An inferior rating is given to cards that do not support any bytecode verification. An adequate rating will be given to cards that support off-card bytecode verification, and a superior ranking will be given to the card that widely supports both off-card and runtime bytecode verification.

1.2.3 Common Criteria/Information Technology Security Evaluation and Certification (ITSEC)
The MULTOS Operating System has been ranked by the Information Technology Security Evaluation Criteria (ITSEC) which is recognized throughout Europe, and the Java Card is rated under the Common Criteria which stems from ISO 15408. Figure 1 shows a table of rankings that are equivalent between the two systems [6].

<table>
<thead>
<tr>
<th>ITSEC</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Criteria</td>
<td>EAL1</td>
<td>EAL2</td>
<td>EAL3</td>
<td>EAL4</td>
<td>EAL5</td>
<td>EAL6</td>
</tr>
</tbody>
</table>

Figure 1. ITSEC and Common Criteria Equivalent Ratings

For the purposes of this study, rankings of EAL1, EAL2 and E1 will be inadequate. Rankings of EAL3, EAL4, EAL5, EAL6, E2, E3, and E4, E5 will be adequate. Rankings EAL7 and E6 will be superior.

1.2.4 Post Issuance Application Installation/Deletion Procedures:
An inadequate rating will be given to a card with no support for digital signatures, no support for encryption of sensitive data and lacks a central authority to regulate and control the application installation process. An adequate rating will be given to a card that supports digital signatures, encryption of sensitive data and possesses a weak governing body that regulates the application installation process. A superior rating will be given to cards with support for digital signatures, encryption of sensitive data, and possesses a governing body of some form that helps ensure the integrity of the card and its applications from card manufacture through the end of life.

All of the ratings above will be measured and tested with the Mann-Whitney rank test for non-parametric data. The significance level is set at 0.05 for rejection of the null hypothesis and the test will be two-tailed.

2. JAVA CARD
The number of Java Cards in circulation is growing at an extremely fast pace. The number of cards issued has increased fifty percent in the last year to 750 million cards worldwide. Sun Microsystems in cooperating with the Global Platform is working diligently to define formal methods and establish the highest security standards in the smart card market today.

2.1 Encryption
The Java Card Application Programmer Interface (API) supports a wide array of cryptographic algorithms, including Advanced Encryption Standard (AES), Data Encryption Standard (DES), Elliptic Curve (EC), Digital Signature Algorithm (DSA), and Rivest, Shamir and Adelman (RSA) encryption algorithm. Sun thus allows Smart Card application designers to choose encryption methods according to the sensitivity of the information being transmitted. Obviously the stronger methods of encryption that use public key technology with a large key size are better suited for protecting cardholder privacy.

The Java Card API supports some symmetric key algorithms (AES and DES) which are older and not the most secure, although they are included in the API for those who wish to use them. Most industrial-strength applications would use public key technologies to provide greater data protection because the private key is never transmitted over the Internet. The Java API most likely includes the symmetric key algorithms for simplicity and convenience for smart card application developers to use if security is not the highest priority.

The RSA encryption technology is industrial strength and can be used to protect the most sensitive data during smart card transactions. RSA is a public key encryption technology that was developed by RSA Data Security. DSA is for the verification of digital signatures and is not an encryption algorithm, although it does use public key technology. The Java Card 2.2.1 API supports RSA keys that are 2048 bits long and DSA keys that are 1024 bits long.

2.2 Java Card Applet Firewalls
Java Card Applet Firewall is an important piece of smart card application security. It provides the framework and the model for controlling access to execution of instructions and to information in the card’s memory. It prevents sensitive information from being leaked from one applet to another [9]. The Java Card Applet Firewall partitions Java packages into contexts separated by firewalls, which enables the card to protect objects, methods and data in other packages from illegal access [9]. Each applet that runs in the Java Card Virtual Machine is associated with an execution context [9]. Each Java package has its own execution context; therefore all applets in the same package share that execution context [9]. There is a one-to-one mapping between packages and execution contexts [9].

Figure 2 shows that the firewall prevents access between applets in different packages. Applets in the same package are free to access each other because the firewall is not between them. As shown in the diagram, not only does each package have its own
execution context, but the Java Card Runtime Environment (JCRE) has its own execution context as well [9].

![Java Card Runtime Environment Context Diagram](image)

**Figure 2. Java Card Firewalls**

While there can be many contexts within a Java Card, only one can be active at any given time. If an object that is a member of one context wishes to access some bytecode of another context, that code will be checked at runtime to determine whether or not a context switch will be made in order to allow access. A context switch would be needed, for example, when an object owned by context A wishes to access a shared object owned by context B, which is a member of a different package [9]. If the context switch is permitted, it results in a new active context that allows the requested access to occur.

Whenever a context switch occurs, information must be stored so that the state of the JCRE can be restored later. Various characteristics of the current context are stored prior to the context switch, so when the method call is completed (after the context switch) the VM can be restored to the condition it was in prior to the context switch [9].

### 2.3 Object Ownership

Every object that exists in the Java Card Runtime Environment is associated with an execution context and also has an owner that is either an applet instance or the JCRE itself [9]. Different applet instances are identified by an application identifier (AID) so the JCRE can distinguish which applet is active in the current context. When executing in an instance method of an object (or a static class method called from within), the object’s owner must be in the currently active context. If not, a context switch must first occur.

Figure 3 is designed to show how object ownership and execution contexts enable the JCRE to determine whether or not to allow access to a particular object’s public methods.

Context switches occur above when method2 calls method3, and a context restore occurs when the call is completed and control returns to method2. No context switch occurs when method1 in Applet A calls a method2 from Applet B. Applet A can invoke methods on Applet B without crossing the firewall and without any context switching. This is because the rules previously established allow access to objects within the same package without a context switch. Notice however, that when the method in Applet B is accessed from Applet A, the object owner changes because we now have a new application identifier while in the method of Applet B.

![Crossing the Firewall Diagram](image)

**Figure 3. Crossing the Firewall**

Consider now that Applet B wishes to invoke a method from Applet C. This requires a context switch because the applet firewall is being crossed and also causes the AID to change. Upon completion of the method call, the context will be restored to that of Applet B and so will the AID.

### 2.4 Object Access

There are special rules for allowing applets to cross the firewall and access other objects. In particular, objects can access objects from other execution contexts only when the objects are Java Card Entry Point Objects, Global Arrays, JCRE Privileged Instructions, or Shareable Interfaces.

Java Card Entry Point Objects are gateways for application objects to request special services from the JCRE [9]. These entry point objects are owned by the JCRE. The applet firewall prevents direct access to the JCRE Objects and methods, therefore the special entry point objects are necessary to allow any execution context to request services from the JCRE. The application firewall prevents access to the fields of these entry point objects, only the public methods are accessible through the firewall [9]. Card applications can request special system services this way, such as the install routine can install a new applet, and the integrity of the JCRE is still maintained by preventing access to fields of objects in other execution contexts.

Global arrays are owned by the JCRE and are accessible from any execution context. For certain types of data it is necessary that it be accessible from any execution context, which justifies the need for global arrays. In order to prevent security issues, references to global arrays cannot be stored in any application classes. Furthermore, global arrays can only be declared by the JCRE, not by card applications.

The JCRE has special privileges because it is a system context. The JCRE can invoke any method of any object in any context at any time. Logically this must be true because at initialization, before any applets are started, the only application running is the JCRE. Therefore, to invoke a Java Card applet, the JCRE must be allowed to kick off the applet. For this reason the JCRE is always allowed to go through the application firewall, forcing the appropriate context switch.

The last inter-applet communication mechanism is the shareable interface, which allows applets of one context to access methods on objects from a different execution context. This can be useful when applets need to provide services to other applets. However, all methods that are publicly available to other execution contexts through a shareable interface must be security-conscious because there is no way to restrict which applets can access them. This
limitation essentially renders the application firewall useless and requires application designers to consider security implications carefully when using shareable interfaces. Considerable precautions must be taken when using shareable interfaces, so that the services being provided are not vulnerable to malicious attacks, or leaking of confidential information.

### 2.5 Code Verification

Applet code that is loaded on to Smart Card devices must be validated to ensure that it has not been corrupted or tampered with prior to installation. Smart card Java applications go through a normal development process where Java source code is compiled into classes. When writing an application for the Java Card, the generated class files are converted into a Converted Applet. The CAP file is generated by a converter which accepts as input the compiled class files and an export file so that the applet can be linked against other packages. The export file contains the names of packages that are being imported by the packages being converted. The installer on the device prepares the CAP file to be loaded into memory so that it can be executed by the Java Card Virtual Machine (JCVM).

One of the major security concerns for any smart card OS vendor is that applet code must be verified to ensure post issuance applications are installed securely without tampering or any malicious attacks. Java Smart Cards have brought to the market many features that were long overdue which are portability, running multiple applications which communicate through shared objects and applets that can be downloaded post issuance [13]. While these are all excellent, they bring on new security issues that must be dealt with by the card operating system. If malicious code is downloaded and installed onto the card, a variety of things could happen, including but not limited to: the leaking of sensitive information such as PINS or cryptographic keys, modification of sensitive information, and causing interference with other applications present on the smart card.

Due to size and memory limitations on the physical smart card itself, the Java Card 2.2 specification provides an off-card bytecode verifier that guarantees the integrity of the CAP file after it is generated and before it is installed on the card. The off-card verifier signs the CAP file with a shared or asymmetric key in order to verify its integrity [11]. When the applet CAP file is installed, it can be decrypted with the key stored on the card to verify that it came from a trusted source (i.e. the card issuer). As technology improves and memory becomes more abundant on smart cards, the move to on-card verifiers will certainly occur [8]. Figure 4 provides a high level overview of the process.

Runtime verification of bytecode has not become widespread due to memory limitations on smart cards. Unfortunately runtime verification increases the overhead for executing each instruction and becomes more memory-intensive for larger programs [1]. The alternate approach proposed by Sun and combines off-card verification with an on-card bytecode verifier. As smart card memory capacity increases, we are likely to see Java Card runtime bytecode verification adopted on a larger scale.

Currently, runtime verification in the Java Virtual Machine implementation consists of the loading of a class file which is then linked, initialized and executed [8]. The linking step includes verification and symbol resolution [8]. When considering Java-Card smart cards, installing a CAP file is equivalent to loading, resolving and initializing. Linking in the Java-Card runtime environment follows an execution path which can contain several variations, although the important note here is that nothing gets executed before it gets verified and resolved.

![Figure 4. Applet installation (post-issuance model)](image)

### 2.6 Java Card Protection Profiles and Common Criteria ratings

There are four protection profiles for smart cards that have been outlined by Sun Microsystems. They are the Minimal Configuration, the Java Card 2.1.1 System Standard Configuration, the Java Card 2.2 System Standard Configuration and the Defensive Configuration [12]. This paper is only concerned with Java 2.2 Smart Card Specifications, we will ignore the Minimal and Java Card 2.1.1 System Standard because it does not support post issuance applet installation, but we do acknowledge its existence.

The Java Card 2.2 System Standard Configuration contains all features of the 2.2 specification which includes the deletion of packages and applets at issuance and post issuance. This specification also includes bytecode verification that is performed by an off-card bytecode verifier [12]. The Defensive Configuration includes everything that the 2.2 System Standard includes as well as an on-card bytecode verifier [12]. This provides maximum security at runtime to reduce the risk of a malicious applet doing harm to card data or other applications.

Defensive platforms are those which contain on-card components which are autonomous with respect to the smart card environment [12]. This includes cards which have an embedded bytecode verifier, and those where bytecode verification is performed at runtime (JSCPP). These cards have sophisticated security architecture in comparison to traditional open smart cards, which is excellent for smart cards where the user can choose to download applications post issuance.

The current Java Card Protection Profile targets the Java Card to have a common criteria rating of EAL4 which means the design has been methodically designed, tested and reviewed [12]. While this is good it means that the designs have not been formally verified and tested, so there is still some question as to the security implications that accompany this rating since the methods have not been proven. Sun Microsystems is working diligently in cooperation with the Common Criteria Organization to firm up these designs and obtain a higher EAL rating for the Java Card.
3. MULTOS

MULTOS is a multi-application smart card operating system that provides a secure environment for smart card applications. MULTOS is composed of two integral parts, the MULTOS Virtual Machine and the Global Key Management Authority (KMA). The virtual machine provides a secure runtime environment for card applications and the KMA offers additional mechanisms which include card enablement, application load and delete certificates and transport key injection, for use by card manufacturers, application developers, issuers, and consumers. The KMA plays a central role in providing a secure environment from chip manufacture, through to the end of the smart card lifecycle.

The MULTOS virtual machine protects applications and data by means of an application firewall. Applications that are written in the MULTOS Executable Language (MEL) contain actions that are processed individually by the virtual machine. Each action to be processed is first checked by the virtual machine to ensure it does not violate security by reading unauthorized areas of memory, executing illegal instructions, or modifying its own code. MULTOS, unlike the Java Card, performs code verification at runtime instead and when the application is loaded onto the card initially. This provides a greater level of security because each and every instruction is being verified every time before it is executed, and the application data is not illegally accessed by other rogue applications on the smart card because of the MULTOS firewall.

Much like the Java Card Virtual Machine, application data can be stored and modified from within each application, as well as in shared data areas. The MULTOS virtual machine provides an API to access and alter this data which is subject to the security restrictions the virtual machine imposes on the applications at runtime.

The other integral part of MULTOS is the Global Key Management Authority. The KMA is the Certificate Authority (CA) for everything related to the MULTOS card, which includes application registration, application load and delete certificates as well as card enablement data. The KMA authority, as well as other processes pertaining to MULTOS, is regulated by MAOSCO which is the governing body for the MULTOS open standard.

Card issuers pay for services from the MULTOS Global KMA, including: registration of the card issuer, registration of applications, generation of card enablement data, generation of application load and delete certificates, and generation of block and unblock data, as explained below. Registration with the KMA allows a card issuer to do business with the KMA. Registered applications are eligible to be deployed to cards issued by a particular card issuer. Card Enablement data makes the card unique to the issuer and sets up permissions and other data that control the operation of the smart card. The application load and delete certificates allow software to be added to and removed from the card post issuance. Finally, the block and unblock data allow the KMA to disable a smart card for any reason, including loss, theft, operational, or any other condition.

The KMA provides transport keys, enablement data and Application load and delete certificates. Transport keys protect the MULTOS cards while in the supply chain, from manufacturer to issuer to consumer. Enablement data assigns the individual card device to a particular card issuer, and can also be used to deactivate the card at any time. Finally, application load and delete certificates give permission to load and delete applications dynamically before card issue and post-issuance.

The KMA uses asymmetric keys, which eliminates the need to transmit the private key. The private key is stored on the card and is used to decrypt information that comes from the card issuer and from the KMA. The private key is specific to that card, which means that only data intended for that particular card can be decrypted. All key pairs for MULTOS cards are created by the KMA and are written to the card device when it is enabled. There is a MULTOS device key that is specific to the card which can be used for various reasons, some of which include securing information about the card issuer and application provider, as well as uniquely identifying the card during enablement and other operations.

3.1 Card Enablement and Application Loading

The MULTOS Global Key Management Authority implements the end-to-end trust architecture. This mechanism focuses on the card issuer’s security, with some key points being [4]:

- The issuer controls which applications and data are loaded on the card.
- Applications loaded on the cards are protected, i.e., no rogue applications can be loaded. All applications must be registered with the KMA.
- The information on the card will not be compromised between actors in the card production process.

Figures 5 and 6 below show the process of post-issuance application installation and deletion.

![Figure 5. Post-issuance application installation](image)

The card elements themselves must be protected during the manufacturing process, passing from the chip manufacturer to a module manufacturer and finally to the card manufacturer. Only then is the card enabled and sent to the consumer. It is important to protect the integrity of the electronic components of the card through all these steps. During chip manufacture an identity (mcd_id) and a transport key are injected into the chip [4] so that the card can only accept data that is specifically targeted for that card.
Once the card has a unique identity and a transport key from the KMA, it can be enabled by the card issuer. The data required to enable the card is created by the KMA, and the loading of the enablement data onto the card is secured by the transport keys created when the chip was manufactured. The process of enablement ties the MULTOS card to a specific issuer and puts the issuer in control of the applications that can be loaded onto the card [4]. For each card or batch of cards, the card issuer provides mcd_id’s, a set of keys used for application loading, its issuer id and product id to the KMA, which in turn supplies enablement data for each card. This information is unique to each card, and can be used at a later time to disable the card if the need arises.

In order for an application to be loaded onto a MULTOS smart card, a few things must be in order. There must be an Application Load Unit (ALU) that has been created which contains the application code, data and other records [4]. The application must be registered with the Global Key Management Authority for use with a particular issuer, an application load certificate (ALC) must be generated from the KMA, the card must receive the ALC, and the information that is contained within the certificate must match the data assigned to the card at manufacture. If all these conditions are satisfied the ALU can be used to install the application on the card (refer to Figure 5 and 7).

Table 1 shows that the MULTOS smart card scores excellent in all four categories. It supports symmetric and asymmetric encryption methods. Code verification is performed off-card and at runtime, and each instruction is checked before it is executed to ensure it is safe and appropriate for execution. The MULTOS card has the second to highest security rating from the ITSEC of E6 which means the protection profile has been formally described, tested and proven to be secure. Finally the MULTOS card has high standards for post-issuance application installation and deletion procedures. The KMA is an independent body that maintains control over the application providers and issues certificates that must accompany the application load unit so the application can be installed on the card.

Table 2 shows that the Java Card scored well, but did not rise to the level of the MULTOS card. The Java Card provides both asymmetric and symmetric encryption mechanisms with asymmetric key lengths up to 2048 bits. The Java Card possesses an off-card bytecode verifier and can verify the code on the card upon installation. Due to memory limitations the Java Card lacks a runtime verification mechanism which results in an adequate ranking. The Java Card has received an EAL4 rating by the Common Criteria which translates into the card has formal specifications that have been defined, but they have not been proven. Therefore the card gets an adequate rating in this category. Finally, the Java Card receives an adequate ranking for post-issuance application installation and deletion. While the Java Card provides secure methods of installing or removing applications by making use of digital signatures, there is no central controlling body that is independent of the card issuer. Therefore the card receives an adequate ranking.

4. ANALYSIS

This section examines the disposition of the hypothesis.
For the Java Card, the statistical value $U_1$ was computed from Table 3. $T_1$ is the sum of the rank scores for the Java Card. $N_1$ is the number of evaluation criteria which equals four.

$$U_1 = T_1 - N_1(N_1+1)/2 = 7 - 4(5)/2 = 7 - 10 = -3$$

For the MULTOS card, a statistical value for $U_1$ was computed from Table 3. $T_2$ is the sum of the rank scores for the MULTOS card. $N_2$ is the number of evaluation criteria, which equals four.

$$U_1 = T_2 - N_2(N_2+1)/2 = 16 - 4(5)/2 = 6$$

The critical value of $U$ obtained with $N_1=4, N_2 = 4$ and a significance level of 0.05 is $U=0$. The lowest value computed is $U= -3$ being lower than the critical value $U=0$, we can reject the null hypothesis with a 95% confidence level and state that there is a significant difference in application security when comparing the Java Card and MULTOS card that did not occur by chance.

5. CONCLUSION

The results above show that the MULTOS card is superior with regard to application security. The MULTOS card has a formal definition and is ranked higher than the Java Card on the ITSEC scale. The MULTOS card’s superior characteristics in terms of runtime bytecode verification, post-issuance application installation and deletion mechanisms, and the strong affiliation of card issuers, application providers, and cardholders with the Global Key Management Authority help to ensure a secure environment for all to do business.

The Java Card has designs in its protection profile to implement an on card bytecode verification mechanism. When this comes to the mainstream consumer, the card will be closer to providing the level of security that MULTOS offers, although it will still not provide runtime verification. As memory on smart cards becomes more abundant it will be easier to implement runtime verifiers that more closely resemble the verification mechanisms the desktop JVM uses on the bytecode it executes. Also object oriented programming in smart cards using Java is very powerful because it is a hugely popular language that most application developers know and using it for smart card applications brings the power of object oriented programming and design to the smart card. Future work needs to be pursued in this area. In addition to this research, further study with regard to improving the runtime bytecode verification algorithms to reduce the overhead associated with these operations so that on-card runtime verification mechanisms can function effectively with a small memory footprint.

Smart cards have come a long way from what they were in previous years. The ability to dynamically load and delete applications is very powerful as this enables card issuers to personalize the smart cards for the cardholder. The ability to quickly and easily enable and disable multi-application smart cards remotely by the card issuer is powerful to prevent information or identity theft.

The difficult work ahead is incorporating smart cards into our daily lives. It is obvious that there are many advantages with smart cards over traditional magnetic stripe cards that offer little or no protection of the data stored on the card. Smart cards offer encryption and biometrics which traditional magnetic stripe cards cannot offer. In addition smart cards provide a secure alternative to magnetic strip cards, which can be consolidated into one secure multi-application smart card. It will not be long before they become a more integral part of our daily lives.

6. REFERENCES


