2014 Eighth International Conference on Software Security and Reliability

High Performance Java Card Operating System

Mohammad R. Eletriby[1], Mohamed Sobh[2], Ayman M. Bahaa-Eldin[3] and Hossam M.A. Fahmy[4]
Computer and Systems Engineering dept.
Ain Shams University
Cairo, Egypt
e-mail: [1]mohammad.eletriby, [2]mohamed.sobh, [3]ayman.bahaa@eng.asu.edu.eg
[4]hossam.fahmy@ieee.org

***Abstract*—Due to the fast evolving of trusted computing environments and internet-of-things an eager need has been established for open platforms which support interchangeable technologies to co-exist without threatening system's security. Certainly, future embedded applications will need high performance operating systems to support the intensive-computing algorithms required for satisfying acceptable response and secure the application inside the vulnerable open environment; hence, new inevitable requirements for embedded operating systems have arisen including hard real-time response, support for native applications, system openness and system scalability. This paper introduces a new design for secure and open smart card operating system, called ESCOS (Egypt Smart Card Operating System), based on the prevalent Java Card technology. The new design provides competitive characteristics in the main three factors of judging smart card platforms; namely, system security, supported technology and system response. In addition, ESCOS is designed to have high degree of modularity and re-configurability to meet fast-changing business needs and diverse hardware platforms.**

Keywords-operating systems; computer security; Java Card; multi-application smart cards; embedded software design; cryptography systems

I. Introduction

Recently a lot of research have been conducted in the area of smart card applications in governmental, financial and military sectors, however, few contributions have been made in operating systems for small cryptographic devices like smart cards/tokens. For years, smart card operating systems have been dominated by research labs of industrial entities particularly the major chip providers. One of the contributions of this paper is to shrink the gap between academia and industry in the field of smart card operating systems and to encourage more public research in this field; hence enrich literature and improve smart card systems development.

The open smart card operating system proposed by this paper depends on Java Card technology the dominating and widely used technology for smart card systems development. Recently, more partners are joining the list of Oracle authorized licensees for Java Card technology. At the time of this paper the list includes more than 30 licensees including the major chip providers; namely, NXP, G&D and STMicroelectronics [11].

Vendors from different sectors including financial, transport and military have recognized the high potential of Java Card technology. For instance, Proton World system the widely adopted ePurse system in Europe, which chose Java Card technology for its operating systems for its high security, flexible functionality and wide availability. On the other side, Java Card technology has influenced telecommunication sector very early by providing a dedicated API for SIM applications starting from Java Card specifications 2.1, since then Java Card has been used as the basic programming technology for GSM systems [4].

Obviously, Java Card has established an eminent position in secure smart card development for its easy application development, leveraging the well-established API specification, which enhances code re-usability and provides high-level interface for application developers. In addition, Java Card is equipped with inherent security features, which include strong memory protection along with applet firewall. Furthermore, Java Card is fully hardware independent, i.e. an applet using Java Card 2.2 can execute in any Java Card 2.2 enabled smart card regardless the underlying hardware. All these advantages together makes Java Card the original way for programming smart cards.

A. Motivation

Despite the numerous benefits of Java Card, the major drawback is the relatively slow execution time particularly for asymmetric cryptographic operation and intensive memory access. This inconvenience is contributed to the interpretation of applet codes. The main contribution of this research work is to provide a prototype for high performance Java Card operating system using a 3-factor enhancing technique. First, the novel operating system layered-architecture, second the enhanced Java Card Virtual Machine design, and last leveraging GlobalPlatform standard to support for native application deployment for very-fast response requirements.

Several remote authentication techniques have been purposed that are hindered by the limited performance of current operating systems. Mainly, these techniques are based on one-way hashing for its low computational cost. ESCOS is developed to realize the feasibility of developing open smart card operating system that provides high security, fast yet secure content management and competitive response especially for security functions; in order to reconsidering high computational cost cryptographic algorithms that provide better security. A smart card operating system with such features will attract application developers to propose new real-time applications for smart cards with higher security prospective. For instance, a full biometric authentication application that securely store biometric information along with processing the matching algorithm on-card. In addition, the proposed operating system will help in the evolution of future SIM cards, remote entrusting technology by using smart card as a Trusted Device, and Identity Management systems for entities authentication using digital identity (e.g. eID and ePassport).

B. *Contributions*

The contributions of this paper include:

- Involving academia in smart card operating system design, which recently has been fully dominated by major chip providers.
- A prototype for open Java Card operating system with significant performance improvement through new layered architecture, enhanced Java Card Virtual Machine design and support for native code deployment.
- A step towards a true general-purpose smart card operating system that can host applications from different technologies.
- Realization of state-of-the-art smart card hardware features including enhanced protocol support, Memory Management Unit, enhanced copy machines and on-board memory encryption.

C. *Smart Card Operating System Requirements*

Recently, smart cards are involved in many public and private sectors applications in governmental, financial and military sectors. Those sectors developed many standards to specify the security and the operation requirements of smart cards. Examples of such standards that describe specific operations and services requirements are: EMV for banking, ICAO for e-passports, CEN/ETSI and GSM for mobile communication, HIPA for healthcare. Concerning security over the system-level and module level the main standards are: FIPS 140(1-3), FIPS 201, CC (EAL 1 to 7). The following standards describe the communication, software and hardware requirements: ISO/IEC 7816, ISO/IEC 14443 and ISO/IEC 15693.

Clearly, those specifications introduce a big challenge for both software and hardware development for smart card operating systems [6], It is required that the smart card operating system supports main standards for communication, security and application management and to pursue international certifications with higher security levels. Probably this will build trust with application providers.

Operating system design should support various communication protocols to facilitate the smart card usage through contact and contactless terminals and should provide flexible, efficient, reliable and secure high level programming language to facilitate application development, along with support for low-level programming to allow vendors to develop complex applications.

In terms of security, the design should protect sensitive data like Keys and PINs against software and hardware attacks. Also, protect applications data in multi-application and multi-vendor environments. While malicious applications are kept isolated, mutually-trusted applications should find a mean of secure inter-application communication if required. For application management security, vendors are allowed to perform secure application download directly while issuing the card or remotely after personalization phase. State-of-the-art security algorithms should be provided to applications with acceptable generation and usage time.

Since one of the main advantages of operating systems is to isolate hardware details from application layer, the design should provide portable software design to reduce effort and cost required for porting to other smart card hardware. At the same time, operating system should fully exploit the advantageous enhanced hardware support provided in modern smart card chips to maximize utilization. The design should be modular and exhibits high-degree of configurability to meet wide spectrum of applications and varying hardware platforms. As smart cards are very limited in memory, smart RAM allocation is required to efficiently utilize the small amount of memory especially in challenging multi-application environment. On the other side, permanent memory should be managed through a common file system to allow secure data storage and sharing.

II. Smart Card Operating Systems Types

Many Smart card operating systems are proposed in the literature in the last 10 years, some of them are educational operating systems like FlexCOS [30], and others are proprietary and commercial. However, to the best of our knowledge, the major available smart card operating systems can be divided into three groups: Global Native, Global Non-Native and Global Mixed.

A. *Global Native Smart Card OS*

This type of operating systems supports GlobalPlatform specifications along with the support for application development using native programming language like C or Assembly. STARCOS [19] is one of the most powerful native operating systems. It allows developers to make high performance applications. However, the major drawback of native development that it requires deep experience with machine programming and may lead to unstable or unsecure applications if not programmed properly. On the other side, smart card operating systems that supports high level programming like Java or MEL is normally supported with many libraries to facilitate application development, in addition to secure framework to protect sensitive data and manage inter-application communication. Developing similar behavior in native applications may lead to better performance and more security, but it requires more development time and more machine programming experience.

B. *Global Non-Native Smart Card OS*

Due to the rapid development of smart card applications, it is required to support high-level languages, which allow fast development, and fulfill the security needs of smart card applications. Two popular operating systems are available: Java based OS and MULTOS.

1) *Java Based Smart Card OS*

This type of smart card operating systems supports verification and execution of Java bytecode according to JCRE specifications early provided by Sun© Microsystems. In 2011 Oracle© announced the release of version 3.0 Classic and Connected editions of the Java Card specifications, where the Classic edition supports automatic garbage collectors, more data and more libraries, furthermore Connected edition allows remote communication and multi-threading [13]. Fig. 1 illustrates the basic architecture of Java Card OS [1].

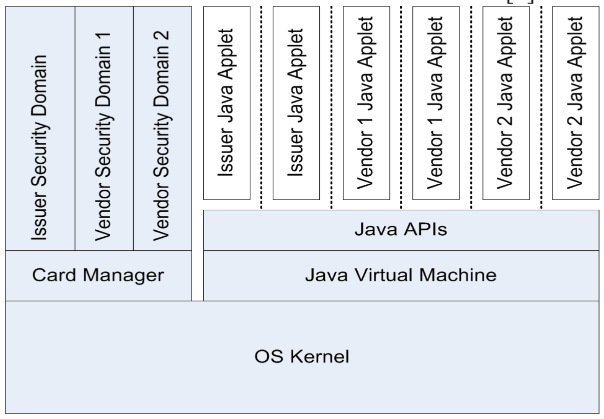

Fig. 1. Java Card architecture

Starting from Java Card 2.1, Java Card allows also for inter-process communication via sharable-interface class. Once the requesting applet (client applet) gets the shareable interface, it can proceed with direct communication with the other applet (server applet). It is a matter of object sharing controlled by the JCRE which, again, is well established in the traditional Java system. However, the mechanism of object sharing in Java Card 2.1 is vulnerable to security attacks by using existing sharable interface to access other sharable interfaces without permission, also, accessing to shareable interface by future applets may be impossible [3].

Java Card supports downloading of custom cryptographic algorithms (developed as Shareable applets) into the system to enable the developers to extend the Java Card API if required. An example of such extension is the implementation of Elliptic Curve Integrated Encryption Scheme (ECIES) over a Java Card v.2.2.1 presented in [32]. However Java language is not efficient in performing complex algorithms like the ones used in cryptographic functions. High performance computing requires native development support, which are not supported by Java operating systems. In addition, Java Card OS does not provide ISO 7816 commands neither internal File System implementation. Alternatively, applet developers should implement the command processing and the File System manipulation if required by their applications.

The most obvious drawback of Java Card OS is the slow execution speed. A rough comparison between a native code and an efficiently-programmed Java applet that implements common smart card commands yields a 30% longer execution time, furthermore the speed can get worse if the applet is not efficiently programmed [4].

2) *MULTOS Smart Card OS*

MULTOS allows developers to use MEL (MULTOS Executable Language), Java, C or Basic languages to develop smart card applications. All supported languages including C are converted to MEL language before downloaded into the card. The OS then executes MEL code using special hardware-independent interpreter. Unlike Java Card OS, MULTOS does not provide multiple separate Security Domains, it depends on strong authentication at application loading, internal isolation and on-chip application conversion and verification. MULTOS itself interferes in the application header signing to allow issuer to install their applications [10].

C. *Global Mixed Smart Card OS*

Both native and non-native operating systems could not provide a complete solution that fulfills all customers' needs. For that, trials are made to introduce a mixed model, which resolves the whole or parts of the problem. In an attempt to reach the mixed model architecture, JCOP 2.4 R2 [16] allows developers to write native libraries and install them via OS provider in the Secure-Box. Secure-Box is a protected user mode internal application where developers can access the new library routines via special Java interface. It provides real native development solution and successfully used by many vendors to add high performance features to their applications like specific security algorithms, accurate fingerprint matching, etc.

Another remarkable attempt towards the mixed model is Caernarvon OS announced in 2008 by IBM® [2,5,8]. The design targets CC EAL7 security level, however, the team succeeded to provide a system prototype and certify the cryptographic module at EAL5+ security level. Caernarvon OS allows developers to write applications in Java or native language like C or assembly and it provides two built-in applications; namely, ISO 7816 and Card Manager. Although Caernarvon OS involves many distinct features, it has also some imperfections. It does not support contactless or USB communication interfaces although contactless is very common and used in many applications since year 2000 [21]. In addition, USB communication is promising in smart card communication due to its high data rates and it can be used to build USB tokens instead of using smart card readers or UART/USB converters.

III. ESCOS OVERVIEW

ESCOS is a smart card OS designed to satisfy all requirements mentioned earlier and to target high security certification level. The system supports both Native and Non-Native (Java) applications, provides modular and hardware-independent design and fully complies with relevant smart card standards and prepared ready for certification process according to the latest Common Criteria evaluation [12]. ESCOS provides state of the art cryptographic techniques including RSA up to 4096 expandable to 8192, also it supports ECC, SHA, AES, DES and RNG. ESCOS supports T=0, T=1 and contactless communication protocol T=CL; while USB support is under development.

ESCOS resolves many security issues by providing onboard Java bytecode verification. For instance, post-issuance application download through unreliable environment can result in deploying malicious applets that can manipulate or destroy other applets on the system. Unfortunately, relying on offcard verifier is not a trusted solution, since the verifier itself could be suspicious or the verified code could be modified after verification; Hence, onboard verifier is the ideal solution. Java firewalls are also provided for the separation between applets. For native applications, they are isolated from the operating system and from other applications using MMU which is responsible for full memory virtualization by translating virtual addresses, used by the processor, into physical addresses. If the memory access operation is invalid, an exception occurs and the execution is directed to the exception handler. By this means, the MMU provides an efficient hardware-based caching and swapping mechanism without degrading the performance, which cannot be achieved by any software implementation.

Moreover, the system provides enhanced hardware features that prevents physical memory attacks. Many recent papers have discussed countermeasures to prevent memory attacks, and it seems that the most promising mechanism is to use lightweight low-latency cryptographic modules, like the one proposed in [31]. This is the countermeasure followed by our design by using the integrity and secrecy of on-chip memory protection module. This module performs encryption of data and addresses for all kinds of the on-chip memory (RAM, EEPROM and ROM) to de-correlate the actual location of data from the logical addresses in memory.

IV. ESCOS ARCHITECTURE DETAILS

Fig. 2 shows ESCOS internal architecture. Before detailing the major components of ESCOS, it is worth noting that the design attains high modularity and re-configurability, where some modules can be excluded from the build according to business needs or hardware constrains. Following modules can be excluded without a need for re-coding: CIU Driver, UART Driver, USB Driver, ISO 7816 T=0/T=1, ISO 14443, JCVM, Cryptographic Algorithms and Built-In Applications. ESCOS architecture is based on following major modules:

A. Kernel

The whole OS consists of three layers; namely, HAL, Kernel, and application layer. Kernel layer features an extensive interface that provides common support for both native and Java applications. The number of layers is intended to be minimized to avoid extra usage of stack since most of smart card hardware provides very small stack size, e.g., NXP P60 provides 128 Bytes stack. If the compiler is configured to use software stack it increases code size and slow down the performance. Even with software stack there is no chance to have deep stack due to the limited memory size (<8KB).

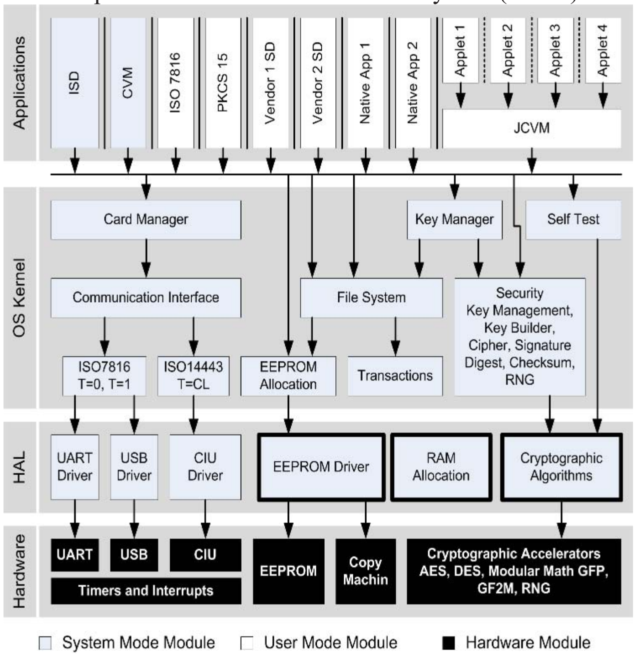

Fig. 2.ESCOS detailed architecture

B. HAL Layer

HAL layer is optimized to minimize the porting cost and to maximize the performance by exploiting the enhanced hardware support provided by the NXP P60 chip. HAL design is very critical as it significantly affects the system performance and portability; for instance, moving kernel operations inside HAL enhances the overall system performance, however it decreases system's portability, so it was essential to use execution profiling tools to determine which low-level operations have the major effect on performance and should be placed inside HAL.

C. Issuer Security Domain

The ISD is a built-in application that is involved in several processes performed by the card that need applying security policy for authentication, integrity and confidentiality, e.g. in downloading and installing of applications this security policy is needed to authenticate the off-card entity and to ensure the integrity of the loaded contents. The ISD is responsible for establishing and terminating of Secure Channels with the terminal device to maintain the confidentiality of the communicated data, if required, through Secure Messaging. The Secure Channel Protocol used in this design is SPC03 with R-MAC/R-ENC support, true random Card Challenge and 16 bytes AES key set [17]. As the design is concerned to be compliant with the standards, ISD is compatible with GP specifications version 2.2.1 with Runtime Messaging support, a feature that enables a selected application to use the services of the ISD on the background [14]. For example, an applet can depend on the ISD to authenticate the external entity and to provide Secure Messaging without implementing its own Secure Messaging protocol; obviously, this saves application code size and simplifies application development.

D. Card Manager

The Card Manager supports all Card Content Management operations (loading, installation, personalization and deletion of applications) for both native and non-native applications, and is designed to minimize the time required for load and install of applications.

Several card evolution mechanisms proposed in the literature use load-time security verification; for instance, the security-by-contract approach for Java Cards in [27,28,29] which proposed a claim checker algorithm at load-time that analyzes the CAP file and matches it with the contract(claim) attached with the downloaded file blocks. indeed this process is time consuming and is not appropriate for the requirements shown at the beginning of this paper, also the claim checker does not provide any solutions for code verification in case of native applications which are more effective in malicious actions since they can directly access the card memory.

On the other hand, without threatening the card security, ESCOS design permits hostile applications to be loaded and installed to the card provided that the application provider can authenticate with the Security Domain. It is the role of the hardware, represented by the MMU, and the system, represented by Java firewall and GP Trusted Framework, to prevent malicious actions regarding memory and inter-application communication. MMU will protect memory access against hostile applications. GP Trusted Framework is responsible for managing inter-application communication, where the requesting application (client application) is checked for illegibility through checking its privileges and the associated Security Domain, if found not illegible, GP Trusted Framework will not grant such communication.

E. Communication Module

The design of CIU, UART and USB modules uses all the enhanced protocol support available in the NXP P60 chip. For instance, the Prologue Buffering and Suppression mechanism is used to suppress T=1/T=CL header of the incoming commands from being received by the receive buffer. As a consequence, the re-transmission of blocks when errors occur is handled completely in the HAL layer, which helped in improving TPDU handling and the receive/transmit buffer is merged into one single buffer to save precious RAM.

For accurate transmission over contactless interface, the design supports a transmission delay mechanism which waits a configurable time after data is written to the transmit buffer, then the transmission starts. This allows setting the CPU in power saving mode during transmission, which significantly reduces the electromagnetic disturbance emissions [15]. The design also supports a Firmware routine for switching the contactless baud rates for reception and transmission during handling of PPS command. The routine guarantees an efficient error-free baud rate switching with maximum supported baud rate of 848 kbps [33]. To increase the speed of communication and increase throughput, Short Guard Time is used for T=1 protocol so each character frame is transmitted in 11 etu instead of 12 etu [18]. This is estimated to save about 10% of the effective data transmission rate [4].

F. JCVM Module

The JCVM is fully responsible for securing the applets while installation or execution via firewall protection. JCVM is built-in native application that can be installed or deleted; also, developers can freely provide other implementations for JCVM or replace it completely with other interpreters like MULTOS or .NET virtual machines. JCVM module is implemented as a software module rather than relying on a Java coprocessor which adds a substantial cost to the hardware platform making Java coprocessor not suitable for our design requirements. Although Java coprocessor dramatically increases execution speed, the coprocessor should be power efficient especially for dual interface cards, since power requirements are so strict in contactless operation. Several papers proposed power efficient Java coprocessor architectures e.g. the one provided in [33]. JCVM internal architecture is provided in Fig. 3. The CAP Converter is responsible for converting the compiled Java code (*.IJC) to special internal representation, to save memory and to speed up the processing. The conversion process is performed externally since it does not affect the security.

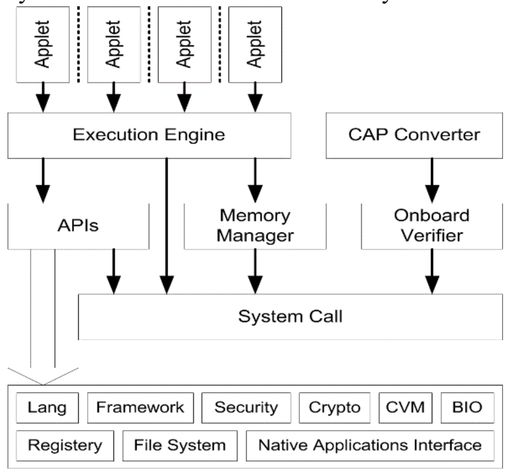

Fig. 3. JCVM internal architecture

The Onboard Verifier is responsible for applying onboard bytecode verification algorithms to check, statically, the correctness of the application's bytecode in terms of syntax and type matching. Since checks are done statically, there is no need for re-checking at runtime, which simplifies JCRE design. Providing onboard verifier will avoid relying on offboard verifiers that might be untrusted, hence emphasizing downloaded applications security. However, onboard verifier has limitations due to high memory usage and high computational cost which are critical factors in smart cards. Many algorithms for onboard verification have been proposed recently to lessen these limitations, for instance the one provided in [20]. Onboard verifier support is mandatory to provide the highest security level according to CC certification standards. On the other side Memory Manager is responsible for allocating transient and persistent memory objects. Java Card uses persistent memory extensively to maintain objects state along the whole card life cycle. When the card is reconnected, Java applets does not need to restore previous objects state because it is already stored in the persistent memory.

G. *File System Module*

ESCOS provides multi-level File System as defined in ISO 7816-4 that supports transparent, cyclic and record-based elementary file structures. The File System uses EEPROM allocation module to allocate memory objects for file headers and files data. For security purpose, the files headers are separated from files data, this is a design issue that is adopted in several embedded File system implementations, for example the SDFS in [25]. The reason behind this separation is to prevent excessive write to data area from flipping the security bits in the headers if they reside in the same memory block. In contrast to SDFS, the resolving of physical memory addresses in ESCOS is done via the MMU since depending on software module in SDFS can't prevent hostile native application from accessing other applications data, in this case the design should imply file verification at load time which is not acceptable in open smart card systems. ISO 7816 is a built-in application introducing the ISO commands for file management. A common File System allows different applications to share the implementation and the data of the File System without the need to re-implement it inside each application.

H. *Transaction Mechanism*

Transaction mechanism is designed to minimize the transaction time and to be compliant with the Java Card specifications. The goal is to support a wide sector of different application requirements especially contactless applications, which require very short transactionntime for that it is more probable to power failures. The transaction mechanism provides its services for both the File system and the JCVM module so that every write operation to a file or a Java object is atomic to ensure data integrity. Although several recent transaction mechanisms for smart cards use transaction buffer caching in RAM, in ESCOS the data is written directly to the EEPROM buffer to minimize the number of memory accesses, hence it fulfills the short transaction time required in contactless applications like automated fare collection and electronic toll collection systems. In addition, the use of fast copy machine in ESCOS reduces the impact of access locality and storage locality of Java objects so no need to use hash tables or logging entries that increase overhead and transaction time [26].

I. *Design Limitations*

HAL layer is fully dependent on the underlying hardware and it is not feasible to support complete portability at this level. The basic operations of smart card hardware are abstracted into extensive interface, which can fit from target to another, however routines implementation would require some porting effort according to the hardware platform.

ESCOS does not support flash memory driver, which is essential when using smart cards as dongole devices with extended flash memory. In addition, RAM compaction is not implemented, so released RAM blocks are not returned to RAM free space. This limitation may affect the total number of multi-selected applications supported through different logical channels. Simultaneous communication with ESCOS through different interfaces (contact and contactless) is not supported. In addition, simultaneous transaction sessions are not supported, i.e. an ongoing transaction should be committed first before beginning a new transaction. The size of transaction buffer is fixed and cannot be expanded dynamically according to data size of the transaction. This limits the transaction space per session, so an application may be required to subdivide a transaction into smaller transaction units. Similarly, file system memory partition is fixed which limits number of applications that can be downloaded into the card. File system cannot be expanded once the card is issued.

Cryptographic algorithms are implemented for Fame2 coprocessors. This limits the portability and certification of the system on other platforms. For instance, when certifying the security module according to FIPS, certification should be repeated for every new port of the security module. Card manager does not support concurrent content management operations requested through different logical channels. In addition, due to memory constraints, number of logical channels permitted to be open simultaneously is 10 channels. Regarding JCVM, the implementation does not support garbage collector.

V. IMPLEMENTATION AND HARDWARE PLATFORM

ESCOS is implemented according to agile iterative process. The implementation considers the following points: stack size is very small, so number of system layers is minimized and recursive calls are forbidden. Transaction operations should be used to update security settings and to manage the card contents, bearing in mind that excessive write access to EEPROM may damage the memory. Secure information, like keys and PINs, should be stored encrypted. Assembly language is allowed only for HAL modules, In addition, the

available hardware, including the enhanced features, should be fully exploited along with supporting alternative software incase hardware is not available for other ports. Internal functions should use structures to pass the parameters. Functions should use workspace structures if the number of local variables is large. It is recommended to reuse buffersand avoid global variables.
The building blocks for ESCOS were mainly the HAL code samples generously provided by NXP to demonstrate the functionality of different components in the hardware platform SmartMX2. These samples are completely re-implemented to meet the new design requirements and the custom interface. In addition, the firmware responsible for adjusting baudrate of contactless interface is used without intervention, since it provides the most stable behavior. All other software modules of the system are completely designed and implemented from scratch bearing in mind CC certification requirements.

To achieve the main security requirements of ESCOS we have to choose the right hardware platform which supports adequate protection and virtualization methods to separate applications from OS and from each other. As discussed in [9], to provide highly secure environment, the hardware platform should provide protection against physical security attacks e.g. power glitching, clock glitching, out of range temperature attacks, differential power analysis and radio frequency leakage. In addition, the platform should provide a fully virtualized memory where the application should be isolated from physical addresses via a translator unit. This memory model can be implemented using MMU that provides a fully virtualized address space for I/O operations. Furthermore, the platform should provide separation between different execution domains (user mode, firmware mode and system mode) to prevent unauthorized access to illegal address spaces; at the same time, there must be a secure mechanism that allows secure transfer of control between the different domains.

The suitable platform to support the previously mentioned features was chosen to be SmartMX2 P60 platform which provides better performance than the previous SmartMX P5 through faster cryptographic coprocessors, more resistant against physical security attacks, more power-efficient design especially for contactless operation, and faster memory module. However, it comes with a factor of 1.3 higher cost than the high end P5 chip. A performance comparison between P60 and P5 chips is presented in the next section.

VI. Performance Evaluation

As the main goal of developing ESCOS is to realize a high speed, secure, and high assurance operating system; the evaluation process followed depends on comparing the response time of ESCOS with the latest operating systems available in market like JCOP family. Response time is chosen as the evaluation criterion, since this is the most convenient method followed by many research papers that compare security operations running in limited-resources platforms like smart cards, e.g. the paper presented in [34]. Fig. 4 shows the testing environment used. Keil IDE with SmartMX2 plugin is used to deploy ESCOS code into the emulation environment along with controlling and monitoring code execution. Emulation environment is based on Ashling SmartICE Emulator for P60 smart card chip. The terminal is simulated using JCOPShell tools operating from Eclipse IDE. The terminal is used to download the testing applets and execute the testing scripts. In case of testing JCOP cards the emulator environment is replaced with JCOP cards.

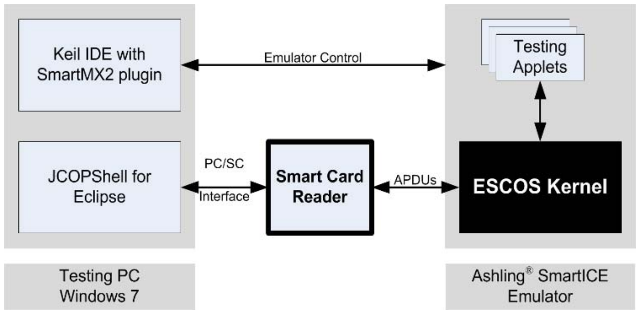

Fig. 4. ESCOS testing environment

The JCOP21 v2.3.1 and JCOP v2.4.1 R3 evaluation cards from NXP were chosen due to their considerable performance and widespread. In addition, they represent the latest smart card technology available to users and they follow the same technology as ESCOS by supporting JavaCard technology and GlobalPlatform.

JCOP and ESCOS use different hardware platforms. JCOP uses SmartMX P5 chip with FameXE cryptographic coprocessor while ESCOS uses the next generation SmartMX2 P60 chip with Fame2. NXP datasheet mentions that the computation performance of P60 is faster than P5, where some computation modules provide faster response up to 3 times while memory transfer module provide faster response up to 5.7 times; cryptographic modules provide faster response up to 5 times [7]. Actually, those numbers describe the extreme performance for some individual operations in best-case scenario, e.g. the fast memory transfer provides maximum speed if the destination is in RAM otherwise the performance is comparable. Table 1 provides the comparison measures.

TABLE 1. PERFORMANCE COMPARISION OF P60 VS P5

	P5	P60	Ratio
Operating Frequency	5 MHZ	5 MHZ	1.0
Instruction Set	8 Bit	8 Bit	1.0
AES Encryption	168 ms	42 ms	4.0
AES Decryption	30 ms	14 ms	2.1
RSA 2048 Sign	612 ms	202 ms	3.0
RSA 2048 Verify	151 ms	92 ms	1.6
4K Copy to RAM	10 ms	2 ms	5.0
4K Copy to EEPROM	55 ms	49 ms	1.1
Pricing (per Unit)	1	1.3	1.3

Before presenting the results of the performed tests, it worth comparing the operating systems according to the supported features as stated in [22,23,24], Tables 2 and 3 summarizes the key features.

TABLE 2. FEATURES COMPARISON (1)

	JavaCard API version	GP version	Chip ID	T = 0	T = 1	T = CL	3DES	AES	RSA
JCOP21 v2.3.1	2.2.1	2.1.1	SmartMX P521	✓	✓		✓	✓	✓
JCOP v2.4.1	2.2.2	2.1.1	SmartMX P5CD080	✓	✓	✓	✓	✓	✓
ESCOS	2.2.2	2.2.1	SmartMX2 P60	✓	✓	✓	✓	✓	✓

TABLE 3. FEATURES COMPARISON (2)

	On-card key generation	Max RSA key	ECC	SHA	H/W RNG	Copy Machine	GlobalPIN	Secure Channel Protocol	PKI co-processor
JCOP21 v2.3.1	✓	2432	✓	1	✓		✓	SCP02	FameXE
JCOP v2.4.1	✓	2048	✓	1, 224, 256	✓		✓	SCP01, SCP02	FameXE
ESCOS	✓	5024	✓	1, 224, 256	✓	✓	✓	SCP03	Fame2

For all of the following performance figures some foundation is considered while performing the test cases, which are represented by the following points:

- The measuring criteria is the response time measured in milliseconds and all timings are represented as averages, where each test is carried out 10 times and the mean value is considered.
- The communication protocol T=1 is used with the maximum baudrate supported by the card under test; however, the communication overhead is ignored, as it is common in both platforms and do not affect the results as it is no more than 10% of the total response time.

A. Card Management Operations

Table 4 shows a comparison between the operating systems under test in performing card management operations. The operations include selection, authentication, application management and card content interrogation. Obviously, ESCOS system outweighs the JCOP systems with about a factor of 4 faster performance in the majority of operations. Furthermore, the application management performance, particularly of loading Applets, is excessively better than its opponents.

TABLE 4. CARD MANAGER OPERATIONS

Card Manager operations	JCOP v2.3.1 (ms)	JCOP v2.4.1 (ms)	ESCOS (ms)
SELECT ISD	21.3	41.3	15.5
INITIALIZE UPDATE	54.14	64.1	18.38
EXT AUTHENTICATE	55.8	66.6	12.87
Put Keyset	78.2	182.9	59.2
Replace Keyset	85.16	166.2	51.3
Installing server Applet			
INSTALL for Load	51.9	77.9	16.2
Package Loading Time	2755.8	3868.4	849.8
INSTALL Server applet	129.2	140.0	80
Installing client Applet			
INSTALL for Load	52.6	138.0	16.4
Package Loading time	1270	1629	283
INSTALL Client applet	127.3	138.2	80.6
Retrieving card information			
GETDATA for ISD and Apps	122.1	151.4	37.4
LOCK App	46.4	54.7	17.3
DELETE Client applet	1320	999.5	110.6
DELETE Server applet	1288	944.7	103.2
Delete Package with Related applets	1335	1000.0	122.3

The performance enhancements of ESCOS in card manager operations is for the following reasons:

- The support of better hardware accelerators and memory management unit.
- The card manager in JCOP family is implemented as an applet, so it suffers longer execution times due to JCVM interpretation, while in ESCOS, card manager is a native application that exploits the full speed of the processor.
- The load file data block hash verification and DAP verification are not implemented in ESCOS since basically ESCOS is developed with the objective of allowing untrusted code, that is potentially hostile, to be loaded into the card and to enforce security between different applications at run time through MMU and the ISD's policy. The load time verification is a time consuming operation since it is applied for each file block to be loaded, consequently ESCOS achieves a substantial gain in speed.
- SmartMX2 CPU implements an address cache mechanism for improved performance of memory accesses. This address cache works on a granularity of 16-byte blocks, i.e. each cache entry covers a 16-byte memory window. If an address is used, which is covered by the cache, the access time will be shorter than accessing an address that is not cached.
- EEPROM features very flexible and fast programming by using a 128-byte page register (intelligent write cache) where it is possible to program 1 to 128 bytes of EEPROM at a time. The bytes which shall be programmed into the EEPROM have to be written first into the page register making sure that only addresses within the target 128-byte page are written to, then a single programming cycle is executed to transfer only

the updated bytes in the page register into the EEPROM, thus EEPROM endurance is increased. The result is a dramatic increase in memory access operations [7].

- The memory compaction in JCOP system is at load-time where the memory is compacted every time a new file is loaded to the system. In contrast, compaction mechanism in ESCOS is done only when no sufficient memory is available for the current file block.

B. *Security Operations*

Table 5 shows a comparison in terms of the response time to symmetric/asymmetric security operations. The operations include key pair generation, encryption, decryption, digital signing and verification. It is worth noting that on-card key generation is a random-based process; thus, the figures given are only average values. However, the key generation function for ESCOS system shows distinct performance especially for large key sizes (2048).

TABLE 5. SECURITY OPERATIONS

Asymmetric operations		JCOP v2.3.1 (ms)	JCOP v2.4.1 (ms)	ESCOS (ms)
RSA key 1024	Key generation	≈2077	≈2579	≈1813
	Encryption	210.3	310.0	182.0
	Decryption	74.5	80.3	73.1
	Sign	299.3	320.8	194.1
	Verify	95.1	125.0	79.3
RSA key 2048	Key generation	≈23565	≈14273	≈5630
	Encryption	863.0	715.3	213.0
	Decryption	107.5	132.1	93.7
	Sign	849.2	729.0	232.0
	Verify	151.4	168.4	104.2
Symmetric operations				
AES key 128	Encryption	200.3	255.3	52.5
	Decryption	45.4	71.1	26.7
AES key 192	Encryption	207.4	253.0	57.3
	Decryption	46.3	77.5	28.1

ESCOS shows better performance for almost all of the security operations. Security operations are largely dominated by the cryptographic software optimization and the hardware speed of the cryptographic coprocessors. Particularly, symmetric operations are completely performed in hardware where the firmware layer provides the blocks handling and the padding schemes. On the other hand, asymmetric operations use hardware to perform primitive modular arithmetic operations only leaving the software to implement the rest of the generation, signing, and verification algorithms beside the block handling and the padding schemes.

C. *Performance Justification*

1) *System Design*

- The SmartMX2 P60 chip provides increased calculation performance of up to factor 3 to existing SmartMX chips, also it provides enhanced orthogonal instruction set that leads to faster execution of commands.
- The JCVM features a new CAP Converter design that reduces the output IJC file size with about 30% compared to regular CAP Converter in JCOP family operating systems. In addition, the new structure of the IJC speeds up reaching the components of the IJC by the execution engine which dramatically increase the JCVM performance.
- The internal architecture of the OS is optimized to provide minimum number of layers, very thin hardware abstraction layer, minimum modules interaction, and very simple and powerful system APIs.
- EEPROM Management is isolated from Transaction and File System modules and all of them are accessible to upper layers modules. This allows fast interaction and prevents redundant sub-modules. In addition, security operations are available via high level and simple interface or low level and direct interface. The low-level interface is used directly by other modules like communication and key management modules.
- The code optimized HAL layer that fully exploits the available features in the underlying hardware provides fast interaction between high-level layers and the hardware.

2) *Hardware Specifications*

- The firewall between applications is provided by hardware module (MMU) transparent to the system, thus it does not put any burden over the card manager.
- Memory encryption is done by hardware to secure the system memory with no load over the system performance.
- Enhanced protocol support for both contact and contactless interfaces that provides fast communication response and decrease the communication overhead. In addition, a CRC/LRC coprocessor that handles frame errors transparently and efficiently justifies the distinct communication performance.
- The improved architecture of the Copy Machine that supports direct memory access to all types of memory including the special function registers of the processor enhances the memory latency and access performance.

VII. CONCLUSION AND FUTURE WORK

ESCOS provides open and scalable operating system complied with standards and depends on the widely used Java Card technology. It allows developers to provide business applications using Java applets, and to provide complex and high performance applications using native programming language. Also, it fulfills all the requirements for secure, high performance smart card operating system. The proposed architecture achieves high performance via novel Card Manager design and enhanced JCVM module, along with employing the state-of-the-art advancements in memory systems and cryptographic coprocessors technologies. On the other hand, ESCOS provides better security; it uses MMU to protect native applications, and uses byte code verification to secure Java applets.

Since one of the main objectives of ESCOS is to reduce the cost of porting, it is essential to realize this feature through providing multiple ports of ESCOS especially for Infineon 16-bit SLE 76 and 32-bit SLE 88 chips in addition to ARM 32-bit Secure-Core chips. On the other hand, more work have to be done to enhance the security of the Java system using on-card modules. Bugs created at the byte code level cannot be detected at run time so it can be exploited by hostile applets to induce a security flaw. Therefore, the use of Byte Code Verifier (BCV) is critical to ensure system security. Our future work include proposing enhanced on-card BCV that has little footprint on the system's performance.

REFERENCES

[1] Jos´e Rafael, Trigueiro de Carvalho, "Comparative analysis of authentication schemes on a Java Card smart card", Master Thesis, Technical University of Lisbon, 2011.

[2] Paul A. Karger, Suzanne K. McIntosh, Elaine R. Palmer, David C. Toll, and Samuel M. Weber, "Lessons learned building the Caernarvon high-assurance smart card operating system", IEEE Security and Privacy Magazine, 2010.

[3] Dieter Gollmann, Jean-Louis Lanet, Julien Iguchi-Cartigny, "Interprocess communication in Java and MULTOS", 9th Volume, IFIP WG 8.8/11.2 International Conference, Springer, 2010.

[4] Wolfgang Rankl and Wolfgang Effing, "Smart card handbook", 4th Edition, John Wiley & Sons, 2010.

[5] Paul A. Karger, David C. Toll1, Elaine R. Palmer, Suzanne K. McIntosh, Samuel Weber, and Jonathan W. Edwards, "Implementing a high-assurance smart card OS", Financial Cryptography and Data Security, Lecture Notes in Computer Science Volume 6052, Springer, January 2010.

[6] Damien Sauveron, "Multiapplication smart card: Towards an open smart card", Elsevier, Information security technical report 14, 2009.

[7] "SmartMX2 family P60x040_052_080 VC", Objective Data Sheet, Rev. 1.1, NXP Semiconductors, 2012.

[8] D.C. Toll, P.A. Karger, E.R. Palmer, S.K. McIntosh, S. Weber, "The Caernarvon secure embedded operating system", Operating Systems Review 42, P. 32–39, 2008.

[9] P.A. Karger, D.C. Toll, S.K. McIntosh, "Processor requirements for a high security smart card operating system" 8th e-Smart Conference, Eurosmart, IBM Research RC 24219 (W0703-091), 2007.

[10] Kenneth R. Wilcox, "Multi-application smart cards: Card operating systems and application security", 21st Computer Science Seminar, 2003.

[11] Oracle, "Java authorized licensees of Java Card technology," 2014. http://www.oracle.com/technetwork/java/javame/javacard/licensees/index.html (accessed April 5, 2014).

[12] "Common Criteria for information technology security evaluation", Version 3.1, Common Criteria, 2012.

[13] "Java card runtime specifications /Virtual machine specifications /APIs reference", Version 3.0, Oracle, 2011.

[14] "GlobalPlatform card specification", Version 2.2.1, GlobalPlatform, 2011.

[15] "ISO/IEC 10373-6: Proximity cards", International Standards Organization, 2011.

[16] "JCOP 2.4.1 product evaluation", NXP Semiconductors, 2010.

[17] "Secure channel protocol 03 - Card specification v.2.2 – Amendment D", Version 1.1, GlobalPlatform, 2009.

[18] "ISO/IEC 7816", 2nd Edition, International Standards Organization, 2004.

[19] "STARCOS S 1.2 reference manual", G&D, 2002.

[20] Berlach, R., Lackner, M., Steger, C., Loinig, J., & Haselsteiner, E. (2014, January). Memory-efficient on-card byte code verification for Java cards. In Proceedings of the First Workshop on Cryptography and Security in Computing Systems (pp. 37-40). ACM.

[21] "ISO/IEC 14443", International Standards Organization, 2001.

[22] "P5Cx012/02x/40/73/80/144 family Secure dual interface and contact PKI smart card controller", Short data sheet, NXP Semiconductors, August 2011.

[23] "NXP J3A080 and J2A080 Secure Smart Card Controller, Rev. 3, Security Target Lite", Evaluation documentation, NXP Semiconductors, December 2010.

[24] "JCOP21 v2.3.1 on secure PKI smart card controller, Rev. 2", Short data sheet, NXP Semiconductors, August 2007.

[25] R. Asgari and R. Ebrahimi Atani, "Secure file management system for Java cards", International Journal in Foundations of Computer Science & Technology, vol. 3, no. 5, pp. 1–11, Sep. 2013.

[26] Yu, Xiaoxue, and Dawei Zhang, "Optimization of transaction mechanism on Java card", In Software Engineering, Business Continuity, and Education, pp. 190-199. Springer Berlin Heidelberg, 2011.

[27] Gadyatskaya, Olga, Fabio Massacci, Federica Paci, and Sergey Stankevich, "Java card architecture for autonomous yet secure evolution of smart cards applications", In Information Security Technology for Applications, pp. 187-192. Springer Berlin Heidelberg, 2012.

[28] Gadyatskaya, Olga, and Fabio Massacci. "Controlling application interactions on the novel smart cards with Security-by-Contract", In Formal Methods for Components and Objects, pp. 197-215. Springer Berlin Heidelberg, 2013.

[29] Dragoni, Nicola, Eduardo Lostal, Davide Papini, and Javier Fabra, "How to secure off-card matching in Security-by-Contract for open multi-application smart cards", In Foundations and Practice of Security: 4th Canada-France MITACS Workshop, FPS 2011, Paris, France, May 12-13, 2011, Revised Selected Papers, vol. 6888, p. 32. Springer, 2012.

[30] Beilke, Kristian, and Volker Roth. "FlexCOS: an open smartcard platform for research and education", In Network and System Security, pp. 277-290. Springer Berlin Heidelberg, 2012.

[31] Ege, Barış, Elif Bilge Kavun, and Tolga Yalçın. "Memory encryption for smart cards", In Smart Card Research and Advanced Applications, pp. 199-216. Springer Berlin Heidelberg, 2011.

[32] Martínez, V. Gayoso, L. Hernández Encinas, and C. Sánchez Ávila. "Java card implementation of the Elliptic Curve integrated encryption scheme using prime and binary finite fields", In Computational Intelligence in Security for Information Systems, pp. 160-167. Springer Berlin Heidelberg, 2011.

[33] He, Junwei, Liji Wu, and Xiangmin Zhang. "Design and implementation of a low Power Java Coprocessor for dual-interface IC Bank Card", In ASIC (ASICON), 2011 IEEE 9th International Conference on, pp. 965-969. IEEE, 2011.

[34] Peng, Zhang, and Jia Jian Fang, "Comparing and implementation of public key cryptography algorithms on smart card", In Computer Application and System Modeling (ICCASM), 2010 International Conference on, vol. 12, pp. V12-508. IEEE, 2010.