Implementation Software to Secure Virtual Machines with Remote Grid of Secure Elements

Hassane Aissaoui-Mehrez, Pascal Urien, Guy Pujolle

▶ **To cite this version:**

Hassane Aissaoui-Mehrez, Pascal Urien, Guy Pujolle. Implementation Software to Secure Virtual Machines with Remote Grid of Secure Elements. IEEE Military Communications Conference (MILCOM'14), Oct 2014, Baltimore, MD, United States. pp.282-287, 10.1109/MILCOM.2014.51 . hal-01092983

HAL Id: hal-01092983
https://hal.science/hal-01092983v1

Submitted on 10 Dec 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Implementation Software To Secure Virtual Machines With Remote Grid of Secure Elements

Hassane Aissaoui-Mehrez & Pascal Urien
Network and Computer Science Department
IMT-TELECOM-ParisTech : LTCI CNRS Laboratory
46 rue Barrault 75634 Paris France
{hassane.aissaoui, pascal.urien}@telecom-paristech.fr

Guy Pujolle
Pierre and Marie Curie University : PHARE
CNRS LIP6/UPMC Laboratory
4 Place Jussieu, 75005 Paris, France
guy.pujolle@lip6.fr

***Abstract*—Security for Future Networks (SecFuNet) is a Brazilian & European research project. The emerging Cloud of Secure Elements infrastructure is used for enforcing identity of Virtual Machines in the Cloud Computing. One of the main goals of the SecFuNet project is to develop a secure infrastructure for virtualized environments and Clouds that not only provides high availability and reliability for users, but that also provides strong isolation among virtual infrastructures.**

The project aims to develop a security framework for Cloud Computing and virtual environments. The goal of this paper is to describe the implementation and the experimentation of the solution for identifying users and nodes in the SecFuNet architecture. In this implementation, only authorized users are allowed to create or instantiate virtual environments. Thus, users and hypervisors are equipped with secure elements, used to open TLS secure channels with strong mutual authentication. Finally, since the physical substrates are shared by several resources (Users, VMs …), the proposed framework must ensure that one resource cannot interfere with the operations of another resource.

Keywords— OpenID; Microcontrollers; Secure Elements; User-Centric Identity; Virtualization and Cloud Computing;

I. INTRODUCTION

While Cloud and virtual infrastructure services can offer great flexibility and convenience for its users, these users no longer have control over the platform on which their services are run. Not only users do not have any guarantees that their services have not leaked any sensitive information, but they may be also subject to attacks by other malicious users in the system. To address this issue, the SecFuNet project proposes to integrate the secure microcontrollers in order to introduce, among its many services, authentication and authorization functions for Cloud Computing and virtual environments. The Identity Management (IdM) system will be based on smart cards and user-centric attribute control policies. The authentication servers are composed by secure microcontrollers. The objective is to implement the framework, based on the authentication servers and EAP-TLS smart card. The proposed SecFuNet framework provides TLS secure channels for establishing trust relationships among Users, VMs, XEN and Remote Grid of Secure Elements (RGoSE). The authentication is done directly between smartcards (owned by users, XEN Hypervisor, or associated to VM) and a RGoSE arranged in a SecFuNet Identity Provider.

This paper concerns a highly secure authentication server with an array of secure microcontrollers allowing VMs' strong mutual authentication with Xen hypervisor. It defines the structure and the components of the authentication server. The first part concerns the definition of the architecture of the server. The second part defines and implements the secure middleware within the Xen server to guarantee a secure communication between VMs and Xen and a response time to an EAP-TLS authentication of few seconds.

This paper is organized as follows. Section II presents a concept of virtual environments and a brief state-of-art of related works on authentication with smart card and introduces the eap-tls smartcard concept. In Section III, we describe the software architecture of Xen Hypervisor and VMs layers. In section IV, we give an overview of the main features supported by our secure middleware in order to handle the remote smart cards plugged in Xen Hypervisor. Next, in section V, we present the experimental results and the performance of our architecture based on a grid of secure elements remotely accessed. Finally, we conclude this paper.

II. CONCEPTS AND PLATFORMS

The SecFuNet framework introduces state-of-the-art technologies to provide an experimental deployment and evaluation of virtual networks taking security as one of the major concerns. To address some of the security issues, the project aims to explore the application of secure elements, such as smart cards, to improve the trustworthiness of network infrastructure services for future networks. Two classes of secure microcontrollers have been studied, smart cards and TPMs (Trusted Platform Modules).

These electronics chips have different computing capabilities : a smartcard [2] is a tamper resistant device, including CPU, RAM and non volatile memory, most of these electronic chips support a Java Virtual machine (JVM) and execute software written in this programming language [3], and therefore are able to execute complex procedures (such as the TLS protocol), while TPMs are dedicated to the RSA algorithm. However these devices may be used in order to enforce trust for the TLS protocol or to guarantee secure

storage for cryptographic keys. These security properties are directly provided by smart cards (thanks to dedicated embedded software), but require additional software components for TPMs.

In this document we call secure element a device such as smart card, which is able to totally or partially handle the TLS protocol and to realize the secure storage or computing of a cryptographic key. The main goal of this section is to briefly overview the context of the virtualization platforms and Grid of EAP-TLS smartcards protocol.

A. Architecture of a Grid of Secure Elements

In this paper we focus on arrays of EAP-TLS smartcards deployed in a Grid of Secure Elements (GoSE).

GoSE is a remote server (i.e. SecFunet Authentication Server, Xen Hypervisor…) hosting a set of secure elements (i.e. smartcards). All accesses to a GoSE require the TCP transport and are secured by the TLS protocol [17] [18] [19].

A GoSE offers services similar to HSM (Hardware Secure Module), but may be managed by a plurality of administrators, dealing with specific secure microcontrollers.

The goal of these platforms is to deliver trusted services over the Internet. These services are available in two functional planes:

- The user plane, which provides trusted computing and secure storage.
- The management plane, which manages the lifecycle (downloading, activation, deletion) of applications hosted by the Secure Element.

Every secure element needs a physical slot that provides power feeding and communication resources. This electrical interface is for example realized by a socket soldered on an electronic board, or a CAD (Card Acceptance Device), (i.e. a reader) supporting host buses such as USB *"Fig. 1,"*. Within a GoSE each slot is identified by a SlotID (slot identifier) attribute, which may be a socket number or a CAD name. The SEID (Secure Element Identifier) is a unique identifier indicating that a given SE is hosted by a GoSE. It also implicitly refers the physical slot (SlotID) to which the SE is plugged.

The GoSE manages an internal table that establishes the relationship between SlotIDs and SEIDs. Therefore three parameters are needed for remote communication with secure element, the IP address of the GoSE, the associated TCP port, and the SEID.

B. EAP-TLS SmartCards Concept

EAP [1] is a universal and flexible authentication framework. Because it can transport about any authentication protocol, it solves the interoperability concerns that their number and their disparity had risen.

Formally, EAP protocol is built on three kinds of messages: requests delivered by servers; responses returned by clients; and notifications issued by servers in order to indicate success or failure of authentication procedures.

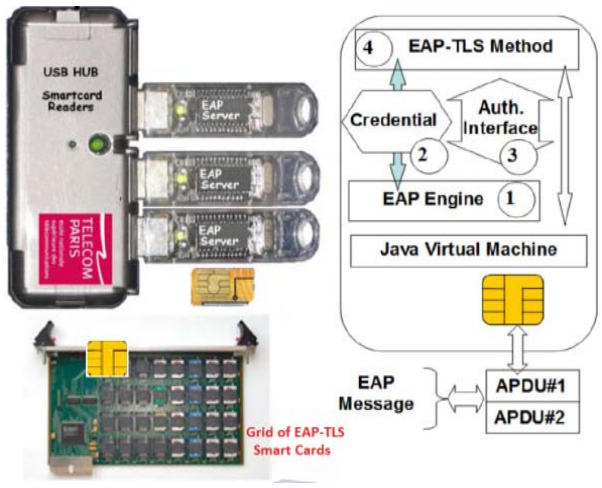
The EAP smartcards functionality and binary encoding interface are detailed in [4]. These devices process EAP methods and act as server or client entity. They communicate via a serial link, whose throughput ranges between 9600 and 230,000 bauds.

There are two classes of operations, sending data (writing to smartcard) and receiving data (reading from smartcard); information is segmented in small blocks (up to 256 bytes) named APDUs, described by the ISO 7816 standard [20].

Security is enforced by multiple physical and logical countermeasures. The use of smartcards in TLS authentication has now a rather long history and has been largely developed according to different models. These devices run the OpenEapSmartcard JAVA open stack, introduced in [5] and which comprises four logical components *"Fig. 1,"*:

- The EAP Engine Object is mostly in charge of the I/O management (i.e. APDUs exchange). It is also responsible of EAP messages segmentation and reassembly. In fact, the APDU payloads maximum length is 255 bytes for input data and 256 bytes for output data, while EAP packets maximum length is about 1300 bytes of data. Consequently, EAP packets are split into several ISO 7816 units, and the Engine entity parses them in order to rebuild the proper EAP packet.
- The Credential Object holds all the credentials required by EAP-TLS method, that is to say: the Certification Authority certificate, the server Certificate and its associated private key. The EAP-TLS State Machine is reset and its according method is initialized with appropriate credentials, each time an EAP-Identity.Response message is received. This object also works as an Identity module. For now, it only holds a unique server Identity but one could possibly load different server Identities issued by several companies which, upon success, would grant different kind of access or services depending of the Client's Identity and its subscription to one or several companies' Network.

Fig. 1. EAP-TLS Smartcard components and GoSE



- The Authentication Interface object implements all services fulfilled by EAP-TLS methods, whose main procedures are initialization, packet processing or MSK key downloading.
- Lastly, the EAP-TLS object is in charge of packets processing, as specified in EAP standard [6]. Since TLS packets size may exceed the Ethernet frame capacity, EAP-TLS supports internal segmentation and reassembly mechanisms.

C. *Virtualization*

More recently, as technologies and applications matured the demand for novel network functionality and requirements began to rise. However, alterations to the Internet architecture have become restricted to simple incremental updates and deployment of new network technologies have become increasingly difficult [7].

This issue became commonly known as "Internet ossification" [8] or "Internet tussle" [9]. In response, network virtualization has been proposed as a means to overcome this impasse by providing diversification to the future inter-networking paradigm [10].

Virtualization is pushing computer systems to new frontiers. From virtualized data centers to cloud providers – scalability, elasticity and multi-tenancy are some of the key required or desired features. This technology has been mostly applied to manage computing and storage infrastructures.

Virtualization techniques create abstract virtual environments composed of a set of "sliced" resources, such as disk, memory, CPU, flow or network. Each virtual environment is isolated and independent from others and thus it seems to run directly over the shared underlying substrate. The virtualization abstraction is accomplished by a software layer called hypervisor, which provides virtual environment interface abstractions quite similar to underlying ones [11].

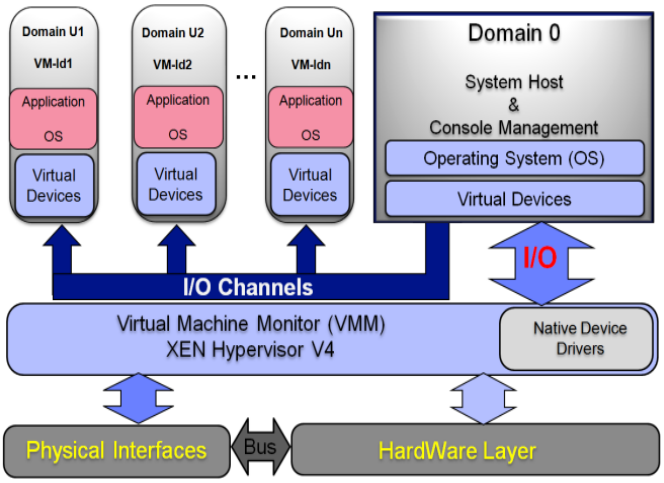
D. *Xen Hypervisor*

The Xen software is a layer called hypervisor, which provides the virtual abstraction, quite similar to underlying physical node. Xen is an open-source that implements a virtualization layer over the physical hardware proposed to run on commodity equipment [12].

The hypervisor creates a virtual environment referred to as Virtual Machine (VM). Each virtual machine accesses a resource abstraction that has its own CPU, memory, disk, and network interface, with its own operating system and applications. The hypervisor controls the access to the hardware and also manages the available resources shared by virtual machines.

A VM is isolated from others, i.e. a running virtual machine is expected not to affect the behavior of other existing virtual machines. Since Xen virtualizes hardware, a VM is an entire computer system, with its own hardware abstraction resources. Xen architecture is composed of one hypervisor located above the physical hardware and several virtual machines over the hypervisor, as shown in *"Fig. 2,"*.

Fig. 2. Xen Hypervisor Architecture



Xen virtualizes the processor by assigning virtual CPUs (vCPUs) to virtual machines and implements a CPU scheduler that dynamically maps a physical CPU to each vCPU during a certain period, based on a scheduling algorithm [13]. Through the paravirtualization technique, Xen enhances guest system performance by changing its behavior to call the hypervisor when necessary. Memory virtualization in Xen is static and the physical RAM is divided between virtual machines. Each virtual machine accesses a fixed amount of memory space, specified at the time of its creation.

In addition, device drivers are kept in a special virtual machine, called the driver domain. The driver domain tasks are often executed by Domain 0, a privileged virtual machine that also executes hypervisor managing tasks.

The driver domain access I/O devices directly by using its native device drivers and also performs I/O operations on behalf of virtual machines.

On its turn, a virtual machine, also called unprivileged domain or Domain U, employs virtual I/O devices controlled by virtual drivers to request Domain 0 for device access [5].

III. Software Architecture Of Hypervisor And VMS

In this section, we detail components composing the infrastructure and platform layer, as illustrated by *"Fig. 3,"*.

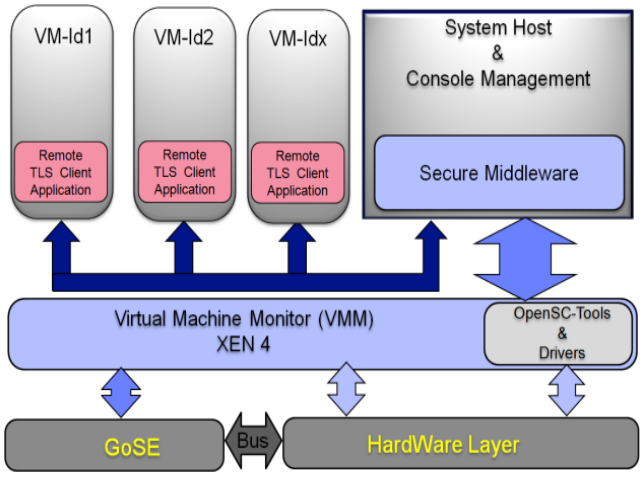
The SecFuNet architecture distinguishes two classes of entities: Nodes and Users. The SecFuNet Users are equipped with two classes of secure elements:

EAP-TLS smartcards or PKCS#1 smartcards, used to open TLS secure channels with strong mutual authentication.

The SecFuNet nodes are divided into three layers of visibility, whose components are identified by a certificate.

The first one (the infrastructure layer) directly uses the services of a secure element, the platform layer and the application layer. Each of them is associated with an administrator in order to manage it.

Fig. 3. Xen Hypervisor And Software Architecture



The infrastructure layer includes hardware resources (servers, secure element grids...) and hypervisors (HV) running in Domain0. Each hypervisor has a certificate, which instantiates its identity. It is associated to a secure microcontroller storing its private key.

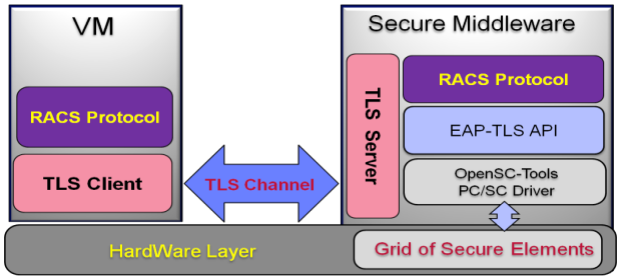
TLS sessions with strong mutual authentication are mandatory for all virtual machines (VM). Thus, we implement the Remote TLS Application in VM-side and the Secure Middleware in Xen-side, in order to enforce a remote VM access to grid of secure elements (GoSE).

A. Secure Middleware API

The Secure Middleware *"Fig. 4,"* is set of software, includes a very simple facility to access GoSE or smart card readers plugged on Xen Hypervisor, with security wrapper mechanisms. This middleware offers some features to secure connections and transactions between GoSE, Xen and VMs. However, the VM may access securely to GoSE and handles the associated remote secure element. The Secure Middleware API is split into several layers:

- PC/SC Driver: talks to GoSE hardware (i.e. readers of Smart Cards). We included the open source project (OpenSC-Tools) [14] for smart card. It implements support for many drivers at the same time. OpenSC is written by an international team and is licensed as Open Source software under the new BSD license.

Fig. 4. Remote TLS Client and Secure Middleware



- EAP-TLS API: is a set of tools and libraries dealing and working with cryptographic capabilities of smartcards. It deals with the credential component that holds all the credentials required by EAP-TLS method (Certification Authority, Server Certificate and its associated private key). The main features of this API are: reset and start the EAP-TLS State Machine of smart card and initialize it with the appropriate identity and credentials.
- RACS: Remote APDU Call Secure protocol for Hardware Secure Module (HSM) Facilities, introduced in [6][15][16]. It provides all the features needed for the remote use of secure elements. The RACS protocol works over the TCP transport layer and is secured by the TLS protocol. One of the main targets of the RACS protocol is to efficiently push a set of ISO7816 requests [4] towards a secure element in order to perform cryptographic operations. In that case a RACS request typically comprises a prefix made with multiple ISO7816 requests and a suffix that collects the result of a cryptographic procedure.
- TLS server: secures connections and transactions between Xen server and VMs. EAP-TLS API launches a TLS server in Xen-side. A TLS Client is running on VM thanks to Remote TLS Client Application. A TLS session is opened between Xen and VM. Then, the Remote TLS Application may handle the remote Smart Card plugged on Xen server using the functionalities provided by EAP-TLS API and RACS. TLS sessions are opened. Upon success, a set of ephemeral cryptographic keys (named KeyBlock) and the list of negotiated algorithms (the CipherSuite parameter) are read from the smart card device.

B. Remote TLS Client Application

Remote TLS Client Application allows a secure connection between VM and Xen and transports ISO7816 commands. When the secure channel is established, the VM can send the APDU commands to smart card via a Secure Middleware API. Sessions with Xen are protected by the TLS protocol with mutual authentication based on certificates. VMs handle the smart card, thanks to RACS Client [6] and EAP-TLS API which hide the encoding of remote procedure call over the networks.

IV. The Functions Supported By EAP-TLS API

The EAP-TLS API consists of number of functions: slot management and cryptographic instructions. This section indicates the main functions that are implemented in the Secure Middleware Library. The main procedures are the following:

- SCard_init_remote_reader: initialize the smart card reader, this function return the number of detected smart card readers, selects the javacard application identity (AID) in smart card device, sets identity and verify code pin, starts a session with a smart card

identified by its index (first index is set to zero), setup the T-Protocol used by a smart card.

- SCard_start_remote_eap: resets the current EAP-TLS state machine and processes an EAP message.
- SCard_connect_remote_reader: opens a connection whit smart card reader.
- SCard_disconnect_remote_reader: closes a session with a smart card identified by its index.
- SCard_list_remote_reader: list all remote smart card readers.
- SCard_send_remote_apdu_cmd: send ISO7816 commands to a smart card.
- SCard_get_remote_cipher_suite: collects the negotiated CipherSuite parameter from smart card device, associated to an opened TLS session.
- SCard_get_remote_key_block: collects the ephemeral keys from smart card device, associated to an opened TLS session.
- SCard_parse_rw_message: read responses from smart card parses it and send messages to VM.
- SCard_process_eap_tls_packet: receives instructions from VMs and make an eap-tls packet to send to smart card.

The Remote TLS Client cans remotely inventory and exchange information with the secure elements. The RACS protocol and EAP-TLS API manage the GoSE or Smart Card and products ease authentication, encryption and digital signatures. This Secure Middleware allows to perform cryptographic operations and access to private key in the smart card associated to VM, and that access is granted only after the VM has been authenticated.

For example: The VM is running in the Domain U of hypervisor. Each VM holds a certificate (that instantiates its identity), but its private key is stored in a secure element plugged in a SIM array (GoSE). All cryptographic procedures dealing with secret keys (such as signing, symmetric encryption or decryption) are performed by the secure element associated to this VM. The protection of asymmetric cryptographic key is a critical issue for the SecFuNet Node.

The authentication between the VM and a Xen server requires the HV private key and VM-Auth-Token (VM Authentication Token). The VM-Auth-Token establishes the proof that the VM (identified by its certificate ID_{VM}) is attached to the HV (identified by its certificate ID_{HV}).

Conceptually the VM-Auth-Token is a signed piece of information (such as certificate) comprising ID_{VM}, ID_{HV} and a validity date; it works with HV public key. Therefore TLS sessions dealing with this certificate imply the use of the HV private key.

VM-Auth-Token = $\{ID_{VM}, ID_{HV}, \text{Validity-Date}\}\text{Sign-Priv}_{HV}$.

V. The Experimental Results

We selected for the SecFuNet project the secure microcontrollers with timings are expressed in ms. Basic cryptographic performances are illustrated by *"Fig. 5,"*:

Fig. 5. Digest costs for the different elementary cryptographic operations.

Blocks to digest	T_{MD5} /Block 64 B	T_{SHA1} /Block 64 B	$T_{RSA\ PUB}$ /Block 128 B	$T_{RSA\ PRIV}$ /Block 128 B	T_{3DES} /Block 8 B	T_{IO} /Byte
Digest Costs /ms	0,49	0,93	25	560	2,10	0,17

- Digest costs (T_{MD5}=*0.49ms and* T_{SHA1}=*0.93ms*). It is the time needed by hash functions to process a block of 64 bytes.
- Symmetric cryptographic costs (T_{3xDES}=*2.10ms,* T_{AES}=*2.60ms,*T_{RC4}=*0.50ms*). It is the time required by symmetric encryption/decryption operations. 3xDES work with 8-byte blocks, while AES deals with 16-byte blocks. It should be noticed that RC4 is a stream cipher algorithm.
- Asymmetric cryptographic costs (T_{RSAPub}=*25ms and* $T_{RSAPriv}$=*560ms*). It is the time required for encryption and decryption with public and private RSA keys (whose size is 128-byte).
- T_{RW} is the time measured by the docking host, to write and read data to/from the non-volatile memory (T_{RW} # 0,17 ms/byte)

Tests were carried out to assess the performance of the developed prototype and attest the infrastructure is viable.

The test configuration is a single server (Xeon Intel Dual Core 3.2 GHz, 12GB RAM), running a Xen hypervisor version 4.

Tow VMs (Fedora 20) were instanced on top of the hypervisor. We do not have access to the actual GoSE, so, the Authentication Server is executed in domain0 with two plugged smart cards in server.

The connection of VM to Xen server is done in three stages:

- Phase 1 is the time to establish secured connection with the Xen server, using tls session between VM and Xen server.
- Phase 2 is the Resume or Full TLS session: the time to establish a connection with the authentication server, using the smartcard associated to the VM. The measurement (T2) is the time between the moment the administrator confirms the PIN and the receipt of the finished message from AS, inside the TLS channel.
- Phase 3 is the time once the secure TLS channel has been established.

Our measures focus on the times of phase 2 and 3 that use transactions with the smart card:

- Phase 2 - Full TLS session: the opening of a full TLS session based on 1024 bits RSA cryptography, requests 2,0s. It exchanges 2500 bytes, whose transfer costs 0,40s. About 230 MD5 and SHA1 calculations are performed (dealing with 64-byte blocks), which consumes 0,32s. One RSA operation with the private key and two RSA procedures with public keys are computed in 0,60s. The sum of the previously cited operations is 1,32s; the remaining time (0,68s = 2,00 - 1,32) is burnt by the java code execution.
- Phase 2 - Resume TLS session: the opening of a resume TLS session costs 0,50s. It requires the exchange of 250 bytes (0,04s), and the calculation of 75 MD5 and SHA1 that consumes 0,11 s. The remaining time (0,35s) is needed by the java code execution.
- Phase III : Once the secure TLS channel has been established, messages exchanged by the client and the server are encrypted with the selected algorithm and integrity is enforced by a HMAC procedure. This phase may either be computed by the EAP-TLS smartcard, or either exported in the docking terminal.

In the case where phase III is processed by the EAP-TLS device, we deduce according to "Fig. 5," that the computing time is mostly consumed by encryption and decryption operations (in other word hash calculations have a second order effect). As an illustration, for a classical RC4, HMAC-MD5 cipher suite, the transfer of 1000 bytes burnt 500ms for encryption, 16ms for MD5 hash, and 170ms for data transmission. Therefore it is mandatory to transfer TLS session from the secure microcontroller to the computer to which it is plugged, in order to obtain an operational data throughput.

VI. CONCLUSION

The architecture proposed and implemented in this deliverable presents a secure solution for VM connection, which aims to develop a highly secure identification scheme, for Cloud Computing, based on the secure elements (smart cards and grid of secure elements). This integration provides a higher level of security when compared to traditional password authentication, and establishes trust relationships among the resources, and eliminates the vulnerabilities like phishing attacks. Strong authentication is done directly between smart cards (owned by user or associated to VM) and a grid of secure processors arranged in a SecFuNet.

We have seen in this paper that the use of secure elements can make a real highlight in the certificate management and the application of a security policy. Adding our Secure Middleware, for large infrastructure, brings the real simplicity of management. In parallel, can be very interesting solution to access securely to a remote specific area of GoSE.

The experimental results of the platform developed for SecFuNet demonstrates that the scalability performances are compatible with today constraints. We are confident, that we will be able to achieve a platform whose authentication time will be reasonable enough to be massively deployed. Furthermore, the smart card shall be a great addition to the Xen architecture and will be as well as a key asset to securing Cloud Computing infrastructures.

Acknowledgment

This work has had financial support from CNPQ through process 590047/2011-6 (SecFuNet project) and also through processes 307588/2010-6 and 384858/2012-0. We also thank CAPES for the financial support with PhD scholarship.

References

[1] RFC 3748, "Extensible Authentication Protocol, (EAP)", June 2004.

[2] Jurgensen, T.M. et. al., "Smartcards: The Developer's Toolkit", Prentice Hall PTR, ISBN 0130937304, 2002.

[3] Chen, Z., "Java CardTM Technology for Smart cards: Architecture and Programmer's (The Java Series) ", Addison-Wesley Pub Co 2002, ISBN 020170329.

[4] IETF draft, "EAP-Support in Smart card", draft-urien-eap-smartcard-25.txt, July 2013.

[5] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in xen," in Proceedings of the annual conference on USENIX '06 Annual Technical Conference, ATEC' 06, (Berkeley, CA, USA), pp. 2–2, USENIX Association, 2006.

[6] P.Urien, 'Remote APDU Call Secure », draft-urien-core-racs-00.txt, August 2013

[7] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," Comput.Netw., vol. 54, pp. 862–876, Apr. 2010.

[8] J. Turner and D. Taylor, "Diversifying the internet," in Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE, vol. 2, pp. 6 pp.–760, 2005.

[9] D. Clark, J. Wroclawski, K. Sollins, and R. Braden, "Tussle in cyberspace: defining tomorrow's internet," Networking, IEEE/ACM Transactions on, vol. 13, no. 3, pp. 462–475, 2005.

[10] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," Computer, vol. 38, no. 4, pp. 34–41, 2005.

[11] N. Fernandes, M. Moreira, Moraes, et al., "Virtual networks: Isolation, performance, and trends," Annals of Telecomm., pp. 1–17, 2010.

[12] Urien, P., "Cloud of Secure Elements, Perspectives for Mobile and Cloud Applications Security", First IEEE Conference on Communications and Network Security" IEEE CNS 2013, 16-19 october 2013, DC USA.

[13] R. Couto, M. Campista, and L. H. M. K. Costa, "Xtc: A throughput control mechanism for xen-based virtualized software routers," in Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, pp. 1–6, 2011.

[14] OpenSC-Tools : https://www.opensc-project.org/opensc

[15] Urien, P., "Cloud of Secure Elements, Perspectives for Mobile and Cloud Applications Security", First IEEE Conference on Communications and Network Security" IEEE CNS 2013, 16-19 october 2013, DC USA.

[16] PKCS#11, https://latinamerica.rsa.com/rsalabs/node.asp?id=2133.

[17] Dierks, T., C. Allen, "The TLS Protocol Version 1.0", RFC2246, January 1999

[18] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006

[19] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", draft-ietf-tls-rfc4346-bis-10.txt, March 2008

[20] [ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO)