SecFuNet : Embedded Framwork in OpenSSL to support Smart Cards

Hassane Aissaoui-Mehrez, Pascal Urien, Guy Pujolle

▶ **To cite this version:**

Hassane Aissaoui-Mehrez, Pascal Urien, Guy Pujolle. SecFuNet : Embedded Framwork in OpenSSL to support Smart Cards. 30TH Annual Computer Security Applications Conference (ACSAC-2014), Dec 2014, New Orleans- Louisiana, United States. hal-01071003

HAL Id: hal-01071003
https://hal.science/hal-01071003v1

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.

SecFuNet : Embedded Framwork in OpenSSL to support Smart Cards

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

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

***Abstract*—Smartcards are becoming increasingly popular as a means for personal identification and authentication in many secure application areas such as e-Banking and e-Commerce. Millions of users have a smart card in their pocket without even knowing it.**

The SecFuNet project proposes solutions for integrating secure microcontrollers in order to develop a security framework for Cloud Computing and virtual environment. This framework introduces, among its many services: authentication and authorization functions for virtual environments, based on Remote Grid of Secure Elements (RG0SE). The objective is to implement an open standard framework, based on smart cards and OpenSSL. This framework provides TLS secure channels for establishing trust relationships among Users, Virtual Machines (VMs), Hypervisor (XEN) and RG0SE. The authentication is done directly between smart cards (owned by users or associated to VM) and SecFuNet Identity Management (IdM).

This framework concerns a highly secure authentication with secure microcontrollers allowing users' (or VMs') strong mutual authentication with SecFuNet Services and provides some libraries to the developers. It defines and describes the features and the modules added to OpenSSL in order to supply easily the Application Protocol Data Unit (APDU) - described by the ISO 7816 standard - transferred to smart cards.

Keywords—Cloud Computing; Virtual Environment; Secure Element; SDK SmartCard; Hypervisor Xen; HSM; OpenSSL; TLS.

I. INTRODUCTION

The SecFuNet objective is to implement an open standard framework, based on the authentication servers and smart cards. The proposed SecFuNet framework provides TLS secure channels for establishing trust relationships among Users, Virtual Machines (VMs), Hypervisor (XEN) and Remote Grid of Secure Element (R-GoSE). The authentication is done directly between Smart Card (owned by users or associated to VM) and SecFuNet Identity Management (IdM).

This paper concerns a highly secure authentication with secure microcontrollers allowing users' or (VM) strong mutual authentication with SecFuNet Services and provides some libraries to the developers. It defines and describes the features and the modules added to OpenSSL in order to supply easily the APDU commands transferred to smart cards.

This paper describes how to integrate a Hardware Security Module (HSM) – EAP-TLS Smart Card – within the OpenSSL tool kit using libraries of Smart Card API on the one hand and the new "s_scard and s_hypervisor" command lines to test connection with SSL/TLS server developed in SecFuNet Project on the other hand. The "s_scard and s_hypervisor" programs may be used for evaluation EAP-TLS Smart Card purposes, for any use. EAP-TLS Smart Card is designed to perform sensitive cryptographic tasks and to securely manage cryptographic keys and data. The security-relevant actions can be executed and security relevant information can be stored. It can be used as a universal, independent security component for heterogeneous computer systems.

The first part of paper concerns the definition of the primitive functions and libraries. The second part describable the implementation of "*s_scard and s_hypervisor"* command lines to use with EAP-TLS Smart Card.

II. CONCEPTS AND STATE OF ART

To address some of the security issues, we 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, smart cards usually run a Java Virtual Machine (JVM) 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 EAP-TLS smart cards services.

A. Eap-Tls Smart Card

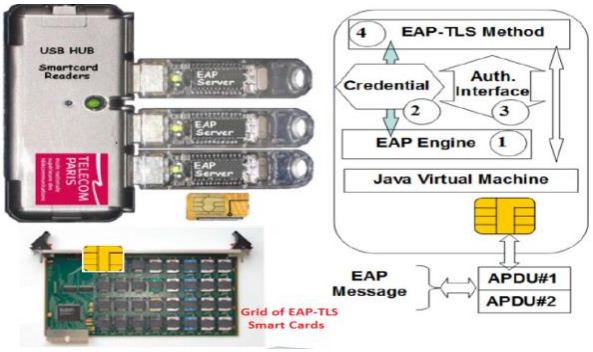
EAP [1] is a universal and flexible authentication framework. Because it can transport about any authentication protocol, it solves the interoperability concerns that their number and their disparity had risen. Formally, EAP protocol is built on three kinds of messages: requests delivered by servers; responses returned by clients; and notifications issued by servers in order to indicate success or failure of authentication procedures.

The EAP-TLS smartcards functionality and binary encoding interface are detailed in [2]. 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 smart card) and receiving data (reading from smart card); information is segmented in small blocks (up to 256 bytes) named APDUs, described by the ISO 7816 standard.

A smartcard [3] is a tamper resistant device, including CPU, RAM and non-volatile memory. Packets exchanged to/from this device are named APDUs and are detailed by the ISO7816 standard. Security is enforced by multiple physical and logical countermeasures. To interface with smart card and smart card reader, we can use PC/SC (short for "Personal Computer/Smart Card") specification for smart-card integration into computing environments without any change. Most of these electronic chips support a Java Virtual machine (JVM) and execute software written in this programming language [4]. 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”:

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.

Fig. 1. The EAP-TLS Smartcard components and grids architecture



2) 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.

3) The Authentication Interface object implements all services fulfilled by EAP-TLS methods, whose main procedures are initialization, packet processing or MSK key downloading.

4) Lastly, the EAP-TLS object is in charge of packets processing, as specified in EAP standard [8]. Since TLS packets size may exceed the Ethernet frame capacity, EAP-TLS supports internal segmentation and reassembly mechanisms.

B. PC/SC Standard

PC/SC is a standard proposed by the PC/SC workgroup which is a conglomerate of representative from major smart card manufacturers and other companies. This specification tries to abstract the smart card layer into a high level API so that smart cards and their readers can be accessed in a homogeneous fashion.

PC/SC is the de-facto standard to interface Personal Computers with Smart Cards (and smartcard readers of course). This high-level and standardized API allows the developer to focus on the smartcard itself, without dealing with various aspects of every smartcard reader. PC/SC is available on Windows integrated in the OS, available through winscard.dll library. Thanks to open-source PC/SC-Lite project, PC/SC is also implemented on numerous UNIX platforms (including GNU/Linux and Mac OS X).

It is formed by a server deamon (pcscd) and a client library (libpcsclite.so) that communicates via IPC. This toolkit was written in ANSI C that can be used with most compilers and does NOT use complex and large data structures such as vectors, etc.

The PC/SC advantage is that applications do not have to acknowledge the details corresponding to the smart card reader when communicating with the smart card.

C. APDU Commands

To handle or invoke smart cards, we use the APDU commands (Application Protocol Data Unit). The APDU is the communication unit between a smart card reader and a smart card. The structure of the APDU is defined by ISO/IEC 7816-4. According to the ISO7816 standards secure element process ISO7816 request messages and return ISO7816 response messages, named APDU. An APDU request comprises two parts: an optional body that may be empty. Its maximum size is 255 bytes and the header is a set of four or five bytes noted:

- CLA indicates the class of the request, and is usually bound to standardization committee (00 for example means ISO request).
- INS indicates the type of request, for example B0 for reading or D0 for writing.
- P1/P2 gives additional information for the request (such index in a file or identifier of cryptographic procedures).
- P3 indicates the length of the request body (from P3=01 to P3=FF), or the size of the expected response body (a null value meaning 256 bytes). Short ISO7816 requests may comprise only 4 bytes.

An APDU response comprises two parts an optional body and a mandatory status word.

- The optional body is made of 256 bytes at the most.
- The response ends by a two byte status noted SW. SW1 refers the most significant byte and SW2 the less significant byte.

D. OpenSSL Project

The OpenSSL Project is a collaborative effort to develop robust applications. The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation. OpenSSL is based on the excellent SSLeay library developed by Eric A. Young and Tim J. Hudson. The toolkit is licensed under an Apache-style license, which basically means that you are free to get and use it for commercial and non-commercial purposes subject to some simple license conditions. This software package uses strong cryptography, so even if it is created, maintained and distributed from liberal countries in Europe (where it is legal to do this), it falls under certain export/import and/or use restrictions in some other parts of the world. OpenSSL is full-featured, and Open Source implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLSv1) network protocols as well as a full-strength general purpose cryptography library and related cryptography standards. The openssl program is a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. It can be used for:

- Creation and management of private keys, public keys and parameters;
- Creation of X.509 certificates, CSRs and CRLs;
- Public key cryptographic operations;
- Calculation of Message Digests;
- Encryption and Decryption with Ciphers;
- Handling of S/MIME signed or encrypted mail;
- Time Stamp requests, generation and verification;
- s_client and s_server commands are two applications Client/Server to test the SSL/TLS protocols.

III. SMART CARDS SUPPORT EMBEDDED WITHIN OPENSSL

The OpenSSL program provides a rich variety of basic cryptography library, certificate services and commands, each of which often has a wealth of options and arguments. In most cases its just used to provide the backend for SSL enhanced protocols like https or ssh. But it also offers a standardized API to other solutions.

The only lack of OpenSSL as a security suite is its tight strings to keys that are stored on a hard disk. This might become a huge security concern, as in most cases anyone with access to the system can just grab the key files and walk away. There are a lot of things an administrator (or a virus with root privileges) can do to steal your key, then the attacker has all time to decrypt the data. That's typically without anyone notifying that the keys and data have been stolen. If the key is on a smart card there is usually no way to export the private key, so if you pull the card from the reader no one can use your keys. And if you use a certified and sealed reader device you can even be reasonably sure that no one can steal your PIN.

The OpenSSL project offers the possibility to source out cryptographic functionality to plug-in specific modules. Usually there is one of two reasons for doing this, performance and security. The performance reason is rather obvious, specialized hardware can do cryptography much faster than a general purpose computer. The reason for using a smart card typically is a security reason. So it is essential to store such keys in a secure manner that requires much more effort to access the keys and ideally disallows the keys to be copied off in an insecure or unattended manner. One solution is to bind OpenSSL to a security device like a TPM, Smartcard, USB Token… or a so called Secure Element (SE) or Hardware-Security-Module (HSM). As a result the key is stored inside such a device and all operations with that key are also done within the secure boundaries. Additional benefits are also, that a key cannot be easily taken away by someone with access to the system, if configured properly it can also not be extracted in any insecure way from the HSM.

Many applications have been written supporting smart cards for various purposes. The most solutions to get the HSM hooked up are based on OpenSSL's engine concept.

However, OpenSSL does not include smart card support. Therefore application developers had to resort to creating their own Software Development kit (SDK) wrappers to access the native PC/SC API for Smart Cards. It is a difficult and time consuming task, time that would better be spent on developing the application itself.

Indeed, some vendors supply their own patches that would require OpenSSL to be modifed and re-compiled. These vendors spend significant resources on developing and maintaining such patches, with the claim that it would be supported.

Our approach is to introduce the SDK Smart Cards API (SDK-SC-API) within OpenSSL that we are developed in SecFuNet project. The SDK-SC-API consists on adding to OpenSSL several primitives in order to handle smart card and to supply packets exchanged to/from the Smart Card device. The SDK-SC-API is a set of procedures written in ANSI C that

can be used with most compilers and for easy access to smart cards from OpenSSL toolkit. This SDK-SC-API implements wrapper functions for accessing GSM SIM and 3GPP USIM cards through PC/SC smartcard library. As this moment, these functions are used to implement authentication routines for EAP-TLS Smart Card based on SSL stack.

These procedures are based on routines specified in winscard like those in the winscard API provided under Windows(R). These procedures are mainly an abstraction of smart cards. It gives a common API for management to most smart cards (EAP-TLS Smart Card, SIM, USIM...) in a homogeneous fashion. Using the SDK-SC-API, an application developer no longer needs to create their own SDK, but rather uses a convenient API that provides a level of abstraction than the native PC/SC API. So instead of patching, modifying and re-compiling your OpenSSL, which might break some system dependencies, make use of standard API's and wrappers.

A. *SDK-SC-API Product Offerings*

SDK-SC-API is not just a wrapper for the PC/SC but provides a framework that can be extended to support any Smart Card authentication (i.e. Using a 2G SIM card. It is described in RFC 4186 from the IETF or using UMTS 3rd generation USIM functionality.

It is described in RFC 4187). Indeed, it is sufficient that smart card reader must comply with the PC/SC standard.

OpenSSL already support third party crypto providers using its engine concept and a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. In our proposition "Fig 2", the additional SDK-SC-API to OpenSSL is available as a source code implemented in C and thus it is managed code in:

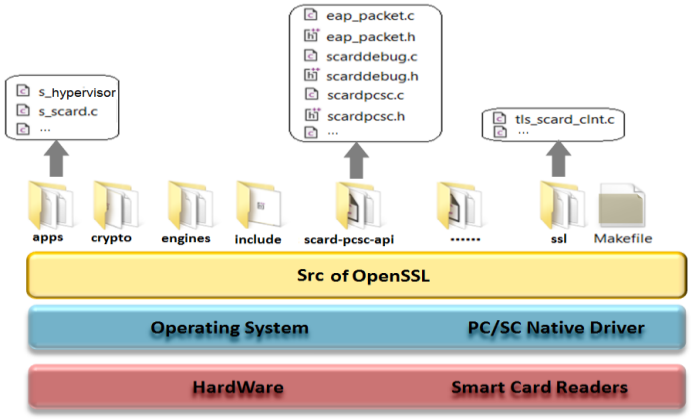
- scard_pcsc_api directory, located in OpenSSL source code, for accessing Smart Cards and managing Smart Cards, this directory contains several files.
- This program provides a test of SSL/TLS protocols with EAP-TLS Smart Card features. It is associated to tls_scard_ctnt.c library dealing with s_scard command to perform EAP-TLS smart card handshake. It is intended for use by any individual involved with integrating smart cards based SSL stack, into an identity and authentication management process. These files are detailed in the next section.

The changes are minor in openssl in order to not impact its structure. Indeed, we have first added two variables in SSL structure:

- SCard_Is_On for indicate to the program (i.e. s_scard) to initialize smartcard reader.
- Reader_nbr for indicate witch smart card reader index to handle.

The second modification consists to modify Makefile in order to build OpenSSL with SDK-SC-API. This SDK has many advanced features, such as support for secure PIN entry. The following list is an overview of the more important files:

Fig. 2. The Additional modification added to OpenSSL



- eap_packet: This file implements all methods that can be used for EAP Full Authenticator state machine with EAP-TLS Smart Cards stack. This state machine is based on the full authenticator state machine defined in RFC 4137.
- scarddebug: supplies methods to manage a reader's events and debug it. These functions are used to print conditional debugging and error messages. The output may be directed to stdout, stderr.
- scardpcsc: contents all procedures and methods which supplies functions to manage readers and to connect to the inserted smart card. Gives the list of readers plugged to the Xen Server and the list of readers having a smart card plugged-in active.
- s_hypervisor: is proxy server, contents all procedures and methods which supplies functions to manage remote readers and associate Smart Card to the authenticated VM. When a VM needs to open a TLS session, the proxy parses all requests (i.e: ClientHello, Certificate Exchange, Client key Exchange…) coming from VM in order to perform this connection using Smart Card associated to this VM.

The SDK module includes sample implementations showing how to achieve complete EAP-TLS Smart Card authentication with an OpenSSL Server.

An Example is making available in the "s_scard.c" program and its "tls_scard_clnt.c" library. The s_scard command is an implementation to test the EAP-TLS Smart Card authentication with an OpenSSL Server.

B. *Authentication With "s_scard" command*

s_scard.c program has been built in a way that makes it easy to customize and modify to suit each developer's requirements. Developers can write and test a custom module according to their own unique requirements.

This program shows how to interact with an OpenSSL Server, how to store keys, and to request TLS/SSL connection. This program can serve as a starting point for developing a

specific module. It should be noted that the "s_scard" command include just an implementation with EAP-TLS Smart Card with an EAP-TLS Smart Card stack "Fig. 1". This program handles locally connected smart card, and can be used for end-to-end testing of TLS client authentication.

As mentioned above, the EAP-TLS Smart Card is an EAP-TLS stack embedded in Smart Card which functions according to the ISO 7816 standard.

For working with EAP-TLS Smart Cards using any PC/SC Smart Card reader, you need make an easy configuration. The use of the PC/SC interface must be enabled and the functions which encapsulate the PC/SC interface have to be adapted so that they can handle the responses from the Smart Card.

IV. Platform Design and Use Case

The platform we have designed for experimental purposes and performances evaluation is represented in "Fig, 3", and is built on three main components:

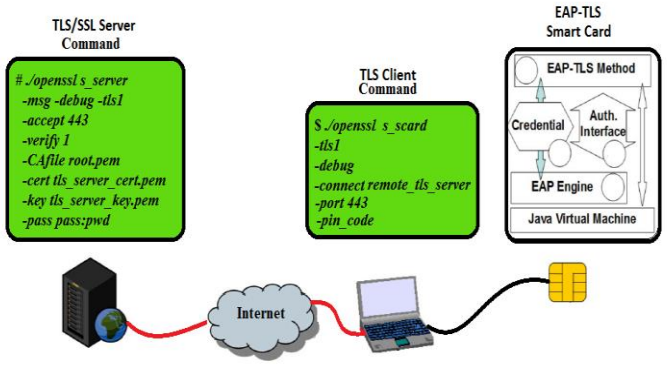
- TLS Server based on OpenSSL, come with a number of tools to facilitate testing, in particular with a TLS/SSL Server environment called "s_server" program.
- EAP-TLS Smart Card device running the OpenEapSmartcard JAVA open stack [5].
- The client host where the Smart Card device is docked and managed by a specific dedicated program. "s_scard.c", which we will call TLS Proxy Client (TLS-PC) for comprehension purposes.

The TLS-PC accesses our EAP-TLS Smart Card, either remotely, or on an internal bus if the TLS server and the TLS-PC are located on the same host. To test the TLS/SSL session with our "s_scard" command line, first, we launch a TLS/SSL Server thanks "openssl s_server" command line.

In our testing example the TLS/SSL Server is running with the following arguments. Ensure that you are familiar with the OpenSSL ToolKit :

#./openssl s_server -msg –debug –tls1 -accept 443 -verify 1 -CAfile root.pem -cert tls_server_cert.pem –key tls_server_key.pem –pass pass:pwd

Fig. 3. The Platform design and experimental of "s_scard" command



In client side, "*openssl s_scard*" command line is running with the following arguments:

#. /openssl s_scard -tls1-debug -connect remote_tls_server:443 pin_code

This command executes the TLS client session between the EAP-TLS smart card and the TLS server on port 443. Note that the TLS Client only needs the remote server name and PIN code.

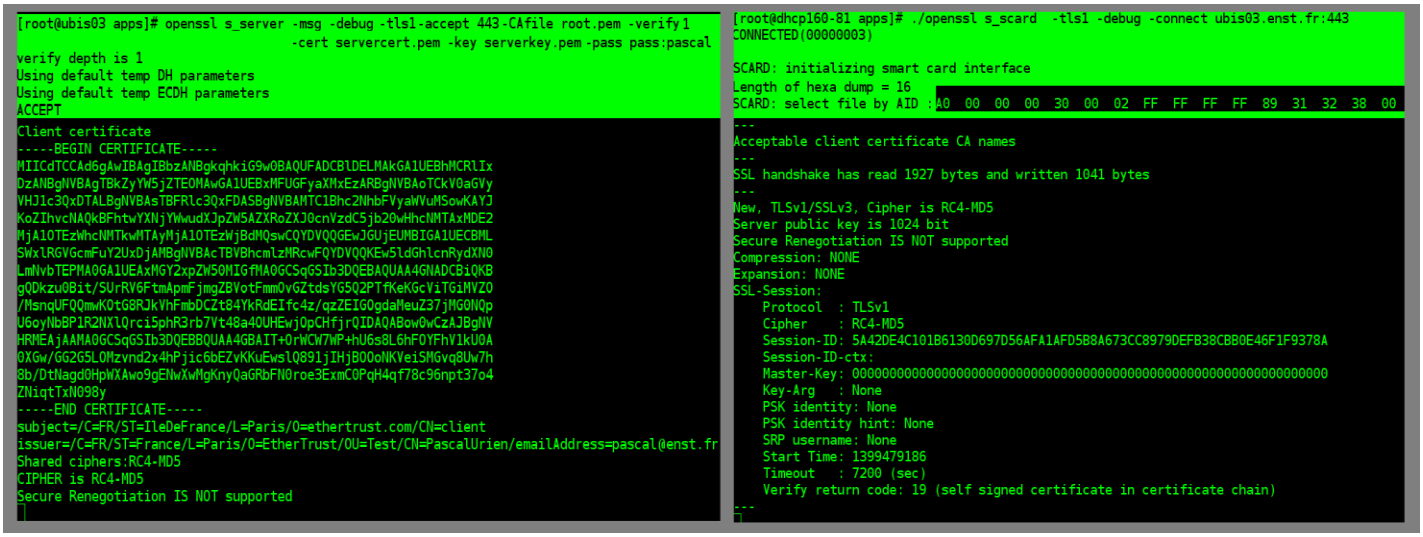
Our TLS-PC program intermediate between TLS Server and EAP-TLS Smart Card to create TLS/SSL connections, whose TLS messages are encapsulated into EAP packets and then, which are forwarded to the EAP-TLS Smart Card thanks to "tls_scard_clnt.c" library dedicated to Smart Cards management.

Once it has been launched, the TLS-PC creates a new connection which will initiate a TLS connection with the TLS Server "Fig, 4". The last server generates a thread on port 443, waiting for socket connections. Each time it receives a connection on this port with EAP-TS Smart Cards according to the main following procedures implemented in "s_scard.c" program and its "tls_scard_clnt.c" library:

- It generates stream sockets connected with the distant TLS Server. Those sockets are used to receive and send the TLS datagram to the TLS Server.
- It checks the datagram, parses it and verifies if it is a proper TLS datagram, or corresponding to an encapsulated EAP message incoming from smart card.
- It splits the EAP message into the appropriate number of APDUs. The EAP message is transported by APDUs thanks to an EAP-Process command, created for that purpose.
- It generates an appropriate context for the APDUs so that they shall be recognized by the Smart Card, which redirects the incoming connection to the proper EAP-TLS Smart Card and whose syntax is specific.
- Upon answer from the smartcard, it parses and reassemblies the EAP packets, in case it has been split by the smartcard into several APDUs, and waits until all the EAP-Request packets have been transmitted.
- It checks the incoming EAP packet from EAP-TLS Smart Card, retrieves the TLS message and encapsulates it into a TLS datagram, and forwards it to the TLS Server.

Once the authentication is successful (or not) and once the keys (key blocs and MSK) have been generated, the Smart Card associated to a TLS session is released and free to be used by a new incoming session. This procedure is renewed as often as necessary until all sessions have been treated and client authenticated.

Fig. 4. The Interaction between *"s_scard"* command and tls server



V. Conclusion

The strong authentication based on smart card has become a critical factor of good system design, it expands as the primary medium for secure communication. In this paper, our contribution consists to add a software development kit "SDK-SC-API" within OpenSSL in order to make easy a development of smart card and to give to developers the possibility to enable a test with smart cards. Furthermore, the security and the advantages it provides shall be a great addition to OpenSSL Toolkits in general as well as a key asset to securing Cloud Computing infrastructures.

We have concluded that the benefits of implementing SDK-SC-API into OpenSSL have many advanced features, such as support for secure PIN entry. The following list is an overview of the more important advantages:

- The client private key is secretly stored and used by the smartcard. The client certificate is autonomously checked by the TLS Server.
- Reduces development time by eliminating the need to create your own SDK wrapper.
- Makes your applications independent of the underlaying smart card devices. The application can be used with PC/SC API and our SDK-SC-API without any change of OpenSSL.
- EAP-TLS Smart Card and (U)SIM can be written on top of SDK-SC-API and can be supported via custom modules.

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

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

[4] D. Recordon and D. Reed, "Openid 2.0: A platform for user-centric identity management," in Proceedings of the Second ACM Workshop on Digital Identity Management, DIM '06, (New York, NY, USA), pp. 11–16, ACM, 2006.

[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] 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.

[7] 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.

[8] P.Urien, 'Remote APDU Call Secure », draft-urien-core-racs-00.txt, August 201