An Innovative Solution for Cloud Computing Authentication: Grids of EAP-TLS Smart Cards

Pascal Urien
Telecom ParisTech
Pascal.Urien@telecom-paristech.fr

Estelle Marie
EtherTrust
Estelle.Marie@EtherTrust.com

Christophe Kiennert
Telecom ParisTech
Christophe.Kiennert@telecom-paristech.fr

Abstract - **The increase of authenticating solutions based on RADIUS servers questions the complexity of their administration whose security and confidentiality are often at fault especially within Cloud Computing architectures. More specifically, it raises the concern of server administration in a secure environment for both the granting access' company and its clients. This paper aims to solve this issue by proposing an innovative paradigm based on a grid of smart cards built on a context of SSL smart cards. We believe that EAP-TLS server smart cards offer the security and the simplicity required for an administration based on distributed servers. We specify the design of a RADIUS server in which EAP messages are fully processed by SSL smart cards. We present the scalability of this server linked to smart card grids whose distributed computation manages the concurrence of numerous authenticating sessions. Lastly, we relate the details of the first experimental results obtained with the RADIUS server and an array composed of 32 Java cards, and demonstrate the feasibility and prospective scalability of this architecture.**

Keywords- Security, Smart card, EAP, TLS, AAA, RADIUS, Cloud Computing

I. INTRODUCTION

Nowadays RADIUS (*Remote Authentication Dial In User Service*) protocol [16] is widely used by Internet Service Providers (ISPs) or companies to grant access to networks services. While it is very difficult to attack or threaten a RADIUS server if it has been properly implemented and secured, it is rather impossible to tell to which extent those who manage this server can be trusted, especially since the server private key is stored on the server machine and can be easily stolen or the exchanges eavesdropped by those who hold the proper rights. This concern is raised by nowadays Cloud Computing technology where the distribution of server can be critical when third-parties are in charge of server management and when no one knows for sure who is in charge of the server's credentials or to which point the person in charges can be trusted, especially in term of industrial espionage. The SSL smart cards grid has been designed to cope with the inherent security issues which are naturally associated to a distributed architecture such as Cloud Computing, since a unique X509 certificate and the SSL stack are securely embedded in each SSL smart card. In fact, smart cards are reputed for the physical security they provide considering it is infeasible to easily access their content or their computing. In addition, the fact that the SSL stack is embedded in the smartcard offers an interesting practicality regarding TLS: would a new TLS flaw be discovered and patched, the smartcards can be conveniently loaded with a more secure and up-to-date TLS stack. Lastly, in term of management, Certificate handling is totally independent from Radius management. As such the server certificates can be updated at will, and the scalability of the server can easily be modulated with a cluster of grids whose number depends of the estimated number of clients. This makes this EAP-TLS smart card grid a convenient paradigm, albeit its performances are reasonable yet far from those of traditional computers.

This paper is organized as follows. Section 2 presents a brief state-of-art of related works. Section 3 introduces the TLS smart card concept. In Section 4, we describe the smartcard enabled RADIUS server. In section 5, we give an overview of the platform design and of its performance based on a grid of 32 cards remotely accessed. Finally, section 6 concludes this paper.

II. STATE OF ART

Most of RADIUS [16] servers support the EAP-TLS [13] [14], i.e. a transparent encapsulation of the TLS protocol [12], working with mutual authentication. Client and server are equipped with X509 certificates and their associated RSA private keys. Our framework merges two innovative technologies: EAP-TLS server smart cards and clusters of such devices, via cloud services.

A smart card [1] 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. Most of these electronic chips support a Java Virtual machine (JVM) and execute software written in this programming language [2].. The use of smart cards in TLS authentication has now a rather long history and has been largely developed according to different models.

Classical frameworks deal with pkcs#15 [3] tokens that store certificates, and compute RSA procedures.

In 2000, a first smart card performing SSL operations was proposed in [4]. However, the weak computing resources of the Java Cards of that time rendered infeasible a full implementation.

Later on, a patent [5] described smart card computing facilities performing functions for TLS exchange, such as certificates checking or signature with private key.

EAP-TLS smart cards, i.e., trusted computing platform running EAP procedures were proposed in 2004 [6], and are detailed later on.

The first grid was designed in [7] and was working with a cluster of java cards. A Mandelbrot set was generated thanks to the combined calculation of smart cards.

Lastly, the use of java cards, processing EAP messages in RADIUS architecture, was previously discussed in [8] [9]. Figure 1 presents the first prototype structure, organized around an USB hub. The RADIUS code is stored in a FLASH disk, and EAP server smart cards are inserted in USB tokens. This component works according to a plug and play paradigm.

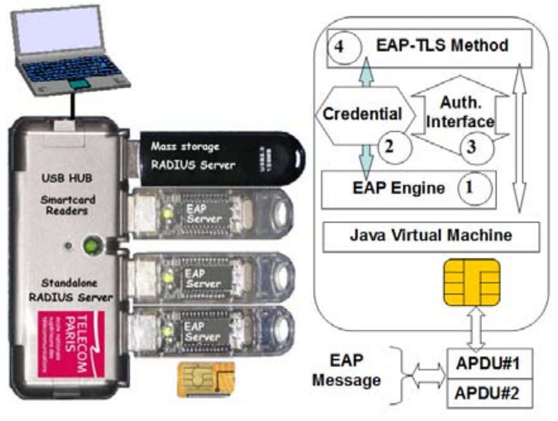

Figure 1. RADIUS server [8], based on EAP-TLS smart cards

In this paper, we propose an architecture that splits the RADIUS server in two parts (see figure 2). First a pure software bloc processes the RADIUS protocol. Second a smart card grid, supports up to four hundred EAP-TLS smart cards, and comprises a mother board and slave extensions, each of them supporting up to 32 smart cards. This electronic rack is usually employed by mobile phone manufacturers who wish to check their compatibility with SIM cards issued by multiple operators. Every smart card is associated with a TCP socket, EAP-TLS procedure is fully managed by a tamper resistant device, and each socket acts as a virtual link used to exchange data with the RADIUS server.

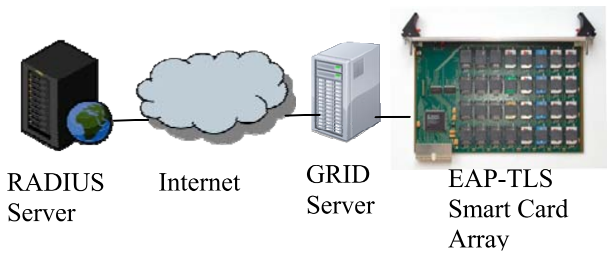

Figure 2. The smart card grids architecture

The main idea behind this architecture is to deploy trust as a service for cloud infrastructures. Each EAP-TLS smart card autonomously processes the SSL protocol. Although the authentication service is distributed over the WEB, critical operations such as mutual authentication (either in full or resume modes) are confined in a trusted computing platform.

III. ABOUT EAP-TLS SMART CARDS

EAP [15] 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 smart cards functionality and binary encoding interface are detailed in [11]. These devices process EAP methods [15] 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.

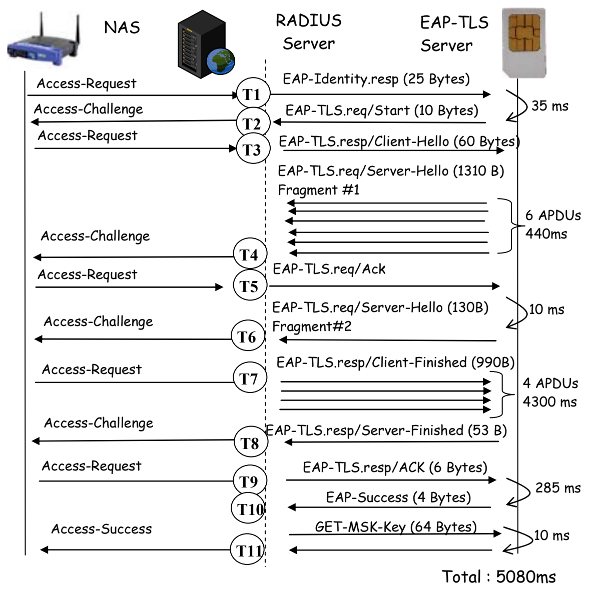

Figure 3. Choreography and timings observed with an EAP-TLS smart card server

In this paper we focus on arrays of EAP-TLS smart cards deployed in grids. These devices run the *OpenEapSmartcard* JAVA open stack, introduced in [10], and which comprises four logical components (see figure 1):

1. The Engine Object is mostly in charge of the IO 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.

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 [15]. Since TLS packets size may exceed the Ethernet frame capacity, EAP-TLS supports internal segmentation and reassembly mechanisms.

Example of computing performances are illustrated in figure 3, the EAP-TLS server runs in the *GX40 java card* manufactured by the Gemalto Company. For convenience, the EAP authentication process is divided in eleven steps; the total procedure costs 5s (with a RSA key size of 1024 bits) and about 2,5 Kbytes of data are exchanged.

IV. About RADIUS

RADIUS technology was developed in the nineties as an access server authentication and accounting protocol, massively deployed in order to solve authentication concerns raised by the increasing number of users who aimed to reach their Internet Service Provider by mean of modems based on PPP protocols. It was then again largely exploited when IEEE 802.1x architecture was introduced, for RADIUS is the key protocol of AAA architecture (Authentication, Authorization and Accounting) and it supports access control mechanisms for wired and wireless infrastructures.

A. Classical Architecture

RADIUS protocol is built on two entities: the NAS or *Network Access* Server which can be a Point of Presence (POP) or an Access Point (AP), and the AS (*Authentication Server*).

In our platform we deal with Wi-Fi infrastructure, compatible with the IEEE 802.1x standard. A wireless client is called a supplicant. Before this supplicant is authenticated and given an IP address, the NAS rejects all frames which do not belong to an authenticated supplicant. For this purpose, EAP authentication messages are exchanged between the NAS and the AS; those messages are transported by LAN or PPP frames and are encapsulated into RADIUS datagrams routed over an UDP/IP stack. To each type of EAP message corresponds a RADIUS datagram (Access-Challenge, Access-Request and Access-Accept / Access-Reject) according to the following scenario:

- The client, or supplicant, tries to access to a network through the affiliated NAS and issues its user's Identity to start the authentication procedure. This Identity is sent by the client terminal thanks to an EAP-Identity message which is then encapsulated by the NAS in a RADIUS *Access-Request* packet and forwarded to the AS. In the case of an EAP-TLS scenario, there is a mutual authentication therefore the user's identity is the subject field of its X509 certificate.

- The AS extracts and analyses the EAP message from the RADIUS datagram and depending on the user's Identity, it will then process the appropriate authentication method. Typically, user's account information and parameters are stored in a LDAP file accessed by the AS, and this information determines which procedure, in our case EAP-TLS, should be initiated to authenticate the user.

- The whole EAP session is then supervised by RADIUS Access-Challenge packets transporting EAP requests, and RADIUS Access-Request packets transporting EAP responses.

- Finally, once the authentication procedure has been finished, the EAP server delivers a notification message, either *failure* or *success*, which is respectively encapsulated in a RADIUS *Access-Accept* or *Access-Reject*. Upon *success*, the EAP server computes a Master Session Key (MSK) which is delivered to the AS through the *Access-Accept* packet. This MSK is both shared by the client terminal and the NAS, and is handled to calculate the session keys needed to encrypt the exchanges between the NAS and the client.

As stated previously, the EAP server is merged within the whole RADIUS module of AS. Most of RADIUS software implementations use the well known *OpenSSL* library in order to support the EAP-TLS authentication procedure, which is a quite transparent encapsulation of the TLS protocol. In our proposal though, EAP server runs in the smart cards and EAP messages are computed by the smart card and forwarded to the AS which then dispatches them to the NAS.

B. Distributed Architecture

We have concluded that the benefits of implementing EAP servers into smart cards are the following:

- The server private key is secretly stored and used by the smart card.

- The client certificate is autonomously checked by the EAP server.

- The SSL stack processed by the smart card is transparent to the RADIUS and the OS in which it has been implemented; it can be easily updated in case of major patches of SSL.

- If the EAP client also runs in a smart card, the TLS stack is channelled from card to card and the EAP session is then fully processed by a couple of tamper resistant devices, working as *Secure Access Module* (SAM), a classical paradigm deployed in highly trusted financial architectures.

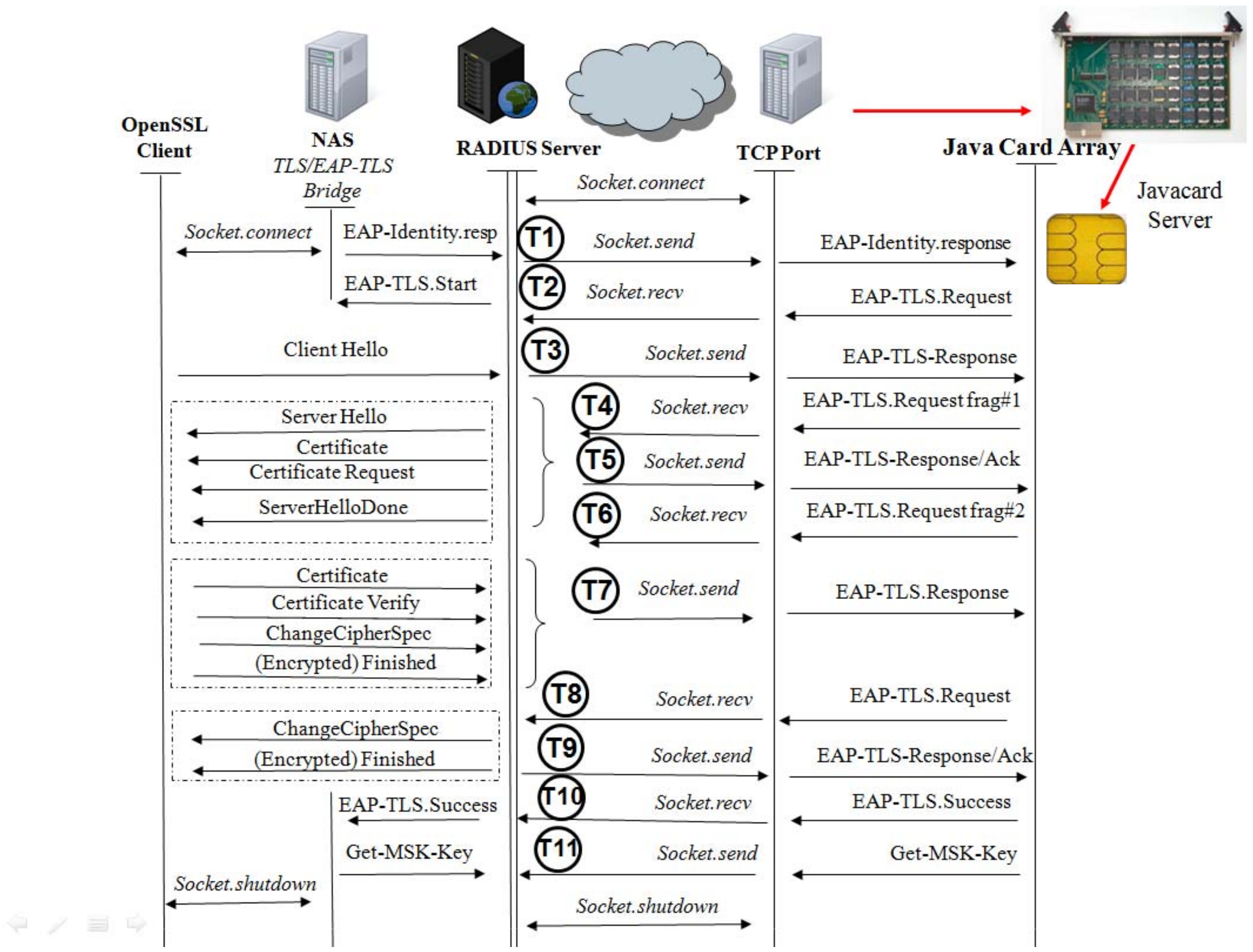

Figure 4. Structure and choreography of the test platform

However, our experimental results demonstrate so far that the performance of our server is much slower than classical RADIUS servers, even if it assures the simultaneous connection of a predetermined number of users.

Our proposed Smartcard enabled RADIUS server is typically a classical RADIUS server which has been split into two main components: a RADIUS authentication server and distributed EAP servers.

The RADIUS authentication server is located on a distant host and is in charge of the following tasks:

- It sends and receives RADIUS datagrams from and to the NAS, thanks to UDP sockets.

- It builds or analyses RADIUS messages and more specifically encapsulates EAP messages from the smartcard into RADIUS datagrams forwarded to the NAS, and reciprocally extracts RADIUS datagrams from the NAS into EAP messages forwarded to the appropriate server smartcard.

- It parses and builds APDUs which are communication units used to interact with the smartcards as explained below.

- It handles the RADIUS secret and computes or checks the associated authentication digest and attributes.

- It opens stream sockets with the smartcards grid and associates an incoming session with a single smartcard and its related connection.

V. PLATFORM DESIGN AND EXPERIMENTAL RESULTS

The platform we have designed for experimental purposes and performances evaluation is represented in figure 4, and is built on three main components: a TLS proxy based on *OpenSSL* and used to simulate a reasonable amount of 802.1x clients as well as an artificial NAS dialoguing with our AS, a RADIUS *Access Server*, and a java card array remotely located and managed by a specific dedicated proxy, which we will call card proxy for comprehension purposes.

The TLS proxy can run up to 30-35 connections, at which point the computer's computation power is not strong enough to assure a decent simulation. It is possible to run this proxy on two different hosts in order to distribute the connections to our RADIUS server, but we have determined that it did not change significantly our results. The SSL proxy accesses our RADIUS server, either remotely, or on an internal bus if the server and the proxy are located on the same host. Our SSL proxy creates a predetermined amount of SSL connections whose TLS messages are encapsulated into EAP packets and then into RADIUS datagrams, which are forwarded to the RADIUS server thanks to datagram sockets directed on port 1812.

Once it has been launched, our RADIUS AS generates a thread on port 1812, waiting for socket connections. Each time it receives a connection on this port, the server creates a new thread which will initiate a connection with one of the server smartcards according to the following procedures (see figure 4):

- It checks the incoming datagram, parses it and verifies it is a proper RADIUS datagram.

- It checks the attribute 79 of the RADIUS message which corresponds to the encapsulated EAP message.

- 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 card proxy, which redirects the incoming connexion to the proper java card and whose syntax is specific.

- It associates a RADIUS session-ID with a specific smartcard so that each incoming TLS session is associated to the same smartcard throughout the whole TLS authentication phase. Once the authentication is successful (or not) and once the keys-blocs and MSK have been generated, the smartcard associated to a RADIUS session is released and free to be used by a new incoming session.

- It generates stream sockets connected with the distant terminal which hosts the card proxy and the java cards array. Those sockets are used to send the APDUs to the remote card proxy and to receive the smartcard response.

- 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 encapsulates the incoming EAP packet into a RADIUS datagram, and forwards it to the TLS proxy located on the client terminal and lastly closes the thread.

This procedure is renewed as often as necessary until all sessions have been treated and all clients authenticated. In case an internal error occurs or in case there is no more java card available, the client’s incoming RADIUS request is silently discarded.

The performance of a single EAP server card linked with a single client has been previously measured [8][9][10]. Another type of java cards was used in our current architecture, but the results we obtained are similar, albeit slightly less efficient. At best, the total cost of an authentication previously measured with a single EAP card directly docked in the server host was about 5000ms, whereas the authentication cost based on the cards we used approximates 6000ms. Now if we initiate the same authentication with the same card located remotely the authentication time is almost doubled. The transfer time has risen drastically, however, since the ping to this remote card proxy is about 30ms, there is an issue here which needs to be farther investigated and fixed in order to obtain reasonable authentication times.

The following measurements have been determined according to the APDU stream. Indeed, the EAP messages are fragmented if necessary, and each EAP-Response packet matches one or several EAP-Process APDUs coupled with the appropriate status words answered by the server card. Those status words indicate that the packets have been properly transmitted or that the server card needs to emit a specific answer which needs to be fetched. Reciprocally, each EAP-request packet matches one or several APDUs coupled with the proper EAP-Response APDU answered by the client card.

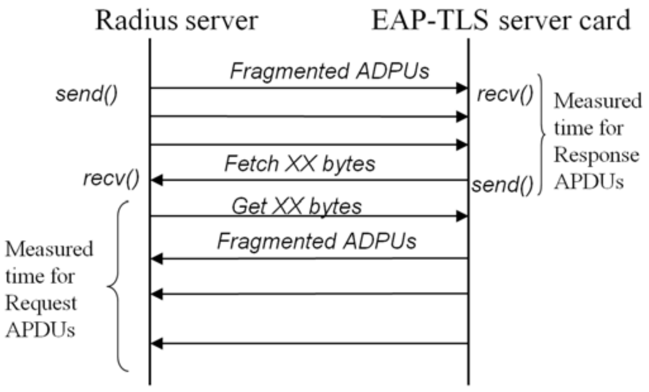

Figure 5. Method for the measurement of transmission time

The transmission time of EAP-Request and Response packets was measured according to the method illustrated by figure 5. Indeed, the time measured for an EAP-Response may be approximated to the time spent between the sending of the APDUs by the RADIUS server, and the reception of the status words sent by the server card. From this point begins the time measurement for the next EAP-Request packet.

	USB Card	Distant Card
T1- Rx: EAP-Identity.response	30ms	100ms
T2- Tx: Start	5ms	60ms
T3- Rx: Client Hello	430ms	580ms
T4- Tx: Server Hello fragment#1	220ms	1850ms
T5- Rx: EAP-TLS-ACK	40ms	100ms
T6- Tx: Server Hello fragment#2	10ms	200ms
T7- Rx: Client-Finished	270ms	2100ms
T8- Tx: Server-Finished	4320ms	4500ms
T9- Rx: EAP-TLS-ACK	290ms	350ms
T10- Tx: EAP-Success	20ms	60ms
T11- Rx: Get-PMK	20ms	190ms
Total	5655ms	10090ms

Tx: EAP-TLS.Request, Rx: EAP-TLS.Response

Figure 6. Timing differences of two EAP sessions established with a USB docked server card or a distant server card.

We will now compare the time measures given by a server card directly docked to the AS terminal with the one given by a distant server card. While a TCP exchange with the distant server can be roughly approximated to 30ms, we observe that the time elapsed to perform a full authentication with the distant server card is greater than expected. In fact, 26 TCP packets are exchanged during a full session and about 2500 bytes are transferred - which we will disregard, considering nowadays broadband data rate. Thus, Tt being the evaluated transfer time, we get:

$$Tt = 26*30 = 780 \text{ ms}$$

In short and at worse, the total authentication time given with a distant card should be one second longer than with a card directly docked to the server terminal, and the results obtained (see figure 6) are far from our expectations.

The total authentication time of a session performed with a USB docked card reaches an average of 5700ms while a session performed with a distant server card takes about 10000ms, which is roughly 3000ms more than expected. Upon investigation of this issue, we noticed that the biggest delays were induced by the largest packets (usually a few hundred of bytes). For instance T7 is very short when the session is performed with an USB server card; however and since it matches the sending of the Client certificate, when the session is performed with a distant server card T7 is ten times longer. In fact the smart grid array works with a 9600 baud throughput for data exchange with smart cards. Therefore, and because about 2500 bytes are required by an EAP-TLS session, these hardware constraints cost 2500 ms. There must be a delay induced with the data processing of the distant proxy in charge of the redirection of the stream to the appropriate server cards. Roughly speaking, an extra delay of about 1000 ms (10000-5700-800-2500) is added by the board *Operating System*.

To confirm this assumption, we tested the scalability and the parallel performance of our distant SIM array with concurrent connections.

	1 Card out of 5	1 Card out of 20
T1- Rx: EAP-Identity.response	220ms	580ms
T2- Tx: Start	100ms	390ms
T3- Rx: Client Hello	580ms	1300ms
T4- Tx: Server Hello fragment#1	2000ms	6300ms
T5- Rx: EAP-TLS-ACK	550ms	2200ms
T6- Tx: Server Hello fragment#2	400ms	1750ms
T7- Rx: Client-Finished	6500ms	21500ms
T8- Tx: Server-Finished	5000ms	6600ms
T9- Rx: EAP-TLS-ACK	350ms	350ms
T10- Tx: EAP-Success	60ms	60ms
T11- Rx: Get-PMK	190ms	200ms
Total	15950ms	41230ms

Tx: EAP-TLS.Request, Rx: EAP-TLS.Response

Figure 7. Average times of EAP-TLS sessions established with distant cards within a 5 or 20 cards concurrency.

When we now start five or twenty simultaneous connections to the remote SIM array; figure 7 shows the average results for one authentication.

Strikingly enough, we can note that the unvarying times such as T9 or T10 match the shortest APDUs, which also means that the parallelisation does work. In addition and as suspected, the certificate exchanges induce an increasing delay (T4 and T7) which prevents the reasonable scalability of this platform. In summary we observe a processing time of 10s for one card, 3s per card for 5 five devices, and about 2s per card for twenty devices.

As of now, we can only establish the fact that there is an obvious issue with the queuing management of the remote server card proxy, which needs to be corrected in order to significantly improve the performance of our SIM array.

VI. Conclusion

In conclusion, although the experimental results of our platform demonstrates that the scalability performances are not yet compatible with today network constraints, we are confident that in a near future we will be able to achieve a platform whose authentication time will be reasonable enough to be massively deployed. Furthermore, the security and practicality it provides shall be a great addition to the 802.1x architecture in general as well as a key asset to securing Cloud Computing infrastructures.

References

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

[2] Chen, Z., "Java Card^{TM} Technology for Smart Cards: Architecture and Programmer's (The Java Series) ", Addison-Wesley Pub Co 2002, ISBN 020170329.

[3] RSA Laboratories, "PKCS #15 v1.1: Cryptographic Token Information Syntax Standard", 2000.

[4] Urien, P., Saleh, H., Tizraoui, A., "SSL in smart card", in proceedings of Journees Doctorales Informatique et Reseaux - JDIR'2000, (Networking and Computer Science PHD days), 6-8 november 2000.

[5] "A Personal token and a method for controlled authentication", Patent# WO 2006/021865.

[6] Urien, P.; Badra, M.; Dandjinou, M., "EAP-TLS smartcards, from dream to reality", in proceedings of Applications and Services in Wireless Networks (ASWN 2004), 2004.

[7] Chaumette S. et. al., "Secure distributed computing on a Java Card grid". 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), 2005.

[8] Urien, P., Dandjinou, M., "Introducing Smartcard Enabled RADIUS Server", The 2006 International Symposium on Collaborative Technologies and Systems (CTS 2006), 2006.

[9] Urien, P., "Open two-factor authentication tokens, for emerging wireless LANs.", Fifth Annual IEEE Consumer Communications & Networking Conference (CCNC'08), 2008.

[10] Urien, P., Pujolle, G., "Security and Privacy for the next Wireless Generation", International Journal of Network Management, IJNM, Volume 18 Issue 2 (March/April 2008), WILEY.

[11] IETF draft, , "EAP-Support in Smartcard", draft-urien-eap-smartcard-18.txt, February 2010.

[12] RFC 2246, "The TLS Protocol Version 1.0", January 1999.

[13] RFC 2716, "PPP EAP TLS Authentication Protocol". October 1999.

[14] RFC 5216, "The EAP-TLS Authentication Protocol", March 2008.

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

[16] RFC 2865, "Remote Authentication Dial In User Service (RADIUS) ", 2000.