A New Convergent Identity System Based on EAP-TLS Smart Cards

Pascal Urien, Estelle Marie, Christophe Kiennert

A New Convergent Identity System Based on EAP-TLS Smart Cards

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

Estelle Marie
EtherTrust
Estel.Marie@EtherTrust.com

Christophe Kiennert
EtherTrust
Christophe.Kiennert@ EtherTrust.com

***Abstract* – This paper presents a new identity system, named SSL-Identity, working with EAP-TLS smart cards. It details an innovative technology based on tiny SSL stacks deployed in billions of such devices embedded in USB dongles or mobile phones, and comments first experimental results. SSL-Identity is convergent because it works in fixed and mobile environment. It claims a strong mutual authentication between user and authentication server. We present the identity lifecycle in which a secure device is manufactured and then linked to one or several user's identities.**

Keywords- Identity Management, TLS, Smart card

I. INTRODUCTION

Identity is the cornerstone for the foundation of trusted and secure services, hosted by WEB servers or cloud computing infrastructures, which may be reached from everywhere at anytime. Although passwords are still widely used for access control to WEB accounts, many applications require strong authentication procedures. PKI infrastructures, in which both server and client are equipped with X509 certificates and associated private keys, can efficiently fulfill this need. However some important issues are not yet solved such as phishing attacks, user mobility on multiple electronic devices like laptops or smart phones.

Moreover, the digital world still lacks a coherent, user-centric identity management system which claims altogether strong security, scalability and high accessibility to end users. In this paper we propose a convergent identity system based on SSL stacks embedded in smart cards. This SSL-Identity model claims a strong mutual authentication between the user and its authentication server. It is a convergent approach since it may be deployed both in USB dongle plugged to laptops or in SIM modules inserted in mobile phones, and also because the associated identity management model can easily be adapted to numerous existing frameworks for identity federation.

This paper details this SSL-Identity model, which lies upon the principle of owning a personal card holding one or multiple user identities. This card is used for authentication with Service Providers thanks to an Identity Federation framework. A former prototype integrating OpenID [6] was developed, providing proof of concept. It was exposed in [13], and earned a Best Demo Award. Additional discussions related to OpenID integration will be detailed in future papers. In this paper, we focus on the identity management part of the architecture, which includes the identity lifecycle going from the emission of the secure device to the revocation of the user's identities.

This paper is organized as follows. Section 2 describes the state of art for identity management proposals. Section 3 presents our technology basis for the building of a convergent model. Section 4 details the identity management model dealing with SSL-Identity embedded in smart card and presents some results obtained with the experimental platform. Finally section 5 concludes this paper.

II. STATE OF ART

Along with the huge increase of services provided to users by websites, which occurred during the last ten years, identity management has become a critical problem which has yet to be solved with satisfying solutions. Be it for online shopping, accessing one's online bank account, or being part of a social network, nowadays users have to authenticate to multiple service providers (SP) [1] in order to access their WEB accounts where sensitive data may be stored. This precisely means that users have to make use of a digital identity, associated to them with credentials, the most common of those credentials being the login/password pair. Since users tend to provide more and more personal data to SP, authentication needs a high security policy which is incompatible with the login/password means.

Moreover, since users have to create an account each time they wish to access a network service, they have to manage multiple identities, and more particularly multiple passwords.

This situation demonstrates that authentication on the Web still misses an unified, convergent identity management system, which could both provide secure authentication and allow the user to perform Single-Sign On (SSO) in a simple way. As of now, many user-centric identity management systems have been developed in response to these authentication issues. Despite being widely deployed and bringing interesting solutions to identity management and SSO problems, they are still mostly unknown by most users.

Identity management is currently provided by technologies such as Windows CardSpace [2], which is a software implementing the InfoCard protocol. Its goal is to give the user the possibility to manage identity cards which can be created personally or delivered by an Identity Provider. The user who wishes to sign on to a SP only has to select a card matching the policy of this SP. The identity does not contain any sensitive information such as a

978-1-4577-0737-7/11/$26.00 ©2011 IEEE

password, which limits the consequences of phishing attacks. Moreover, mutual authentication is performed with the SP, and the identity is sent through an encrypted token. However, the InfoCard protocol is not yet as widely deployed as it could have been, and still not many websites accept this authentication means. Despite its qualities and security strengths, the InfoCard protocol suffers from portability problems, since the identity cards are stored as XML files on the user's computer. This later issue is also another potential flaw in the protocol, as data stored on a computer can never be considered as secure.

Identity management had been anticipated by older projects, developed about ten years ago, that aimed to provide federated identity within a circle of trusted websites. The most popular ones were Microsoft .NET Passport, Shibboleth [3] and Liberty Alliance [4], but the .NET Passport project did not meet the expected success and currently only focuses on Microsoft services such as MSN or Xbox. As for Shibboleth, it made use of technologies very similar to those of Liberty Alliance, but mainly focused on university contexts, where it is largely in use as of now.

However, Liberty Alliance currently holds a major place in the identity management world. With a membership of more than 150 organizations, Liberty Alliance aims to widely deploy federated identity infrastructure with technologies based on SAML [5] and adapted to Web services. However, most users are still not familiar with Liberty Alliance solutions because of the complexity of the project. Moreover, SSO solution remains mainly implemented by the classic login/password pair, which the user has to give to the identity provider. Therefore, Liberty Alliance is not yet able to offer user-centric identity management and simple means to perform SSO within a circle of trusts.

A user-centric solution to federated identity issues has been proposed by the emerging technology OpenID [6], where a user only has to create an account in the OpenID identity provider of his choice, and then gets a XRI (Extensible Resource Identifier) identifier which he can use to perform SSO authentication to trusted websites. The authentication phase redirects the user to the OpenID Identity Provider server, where he needs to enter his credentials, before being redirected to the SP as an authenticated user. Despite its very simple and efficient architecture, OpenID still suffers from several security flaws. First, OpenID mainly stands on a password-based authentication, which remains a major flaw in itself. Moreover, SSL server authentication is not mandatory, which allows phishing attacks to be performed in order to get the user OpenID password. More generally, OpenID implementation needs to be coupled to strong authentication means in order to provide satisfactory security.

In section IV, we propose such a solution based on smart cards embedding SSL. As explained in section III, smart card based SSL authentication is considered as a strong authentication means and can be applied with many benefits to SSO authentication. Moreover, frameworks such as OpenID have no specific recommendations concerning identity management, which makes the model presented in section IV particularly appropriate as a complement to open federated identity frameworks.

III. CONVERGENT IDENTITY TECHNOLOGY BASIS

A. Smart cards

A smart card [8] is a tamper resistant device. It is a low cost secure micro-controller that comprises a CPU (8 to 32 bits design), ROM (a few hundred KB) hosting the chip operating system, RAM (about 10 KB) used for stack operations, and non volatile memory (from 32 KB to 1MB) storing software and cryptographic keys. It communicates via a serial link or an USB interface; the data throughput ranges from 9600 Kbits/s to a few Mbits/s; the packets exchanged via this communication facility are called APDUs (*Application Protocol Data Units*), according to the ISO 7816 standard, and their size is about 256 bytes.

Security is enforced by various software and hardware countermeasures. Software implementations of cryptographic algorithms are subject to multiple attacks such as faults injection able to recover hidden keys from computing errors. Physical effects induced by various processors components (buses, gates, charge switching...), usually referred as *side channel effects* may also be used to crack a cryptographic secret. Because most of computers are built from untrustworthy hardware components, Trojan horses and other malicious software can hack cryptographic credentials.

Most of smart cards are equipped with a virtual machine, either a Java Virtual Machine or a .NET interpreter. Consequently critical software components easily run in these trusted computing platforms; in this paradigm, security is a program.

Basically there are two classes of identity procedure; first is fully managed by its owner (a password is a perfect example of this concept), second is controlled by a third party. This last case is typically illustrated by USIM smart cards issued by 3G mobile operators, deployed for authentication over radio networks but whose bearers ignore the secret key value embedded in the secure chip. A convergent identity model works for the two modes previously mentioned; the proof of identity is not performed by non trusted components, which avoids identity theft or hijacking of legitimate subscribers.

The TLS protocol is the holy grail of the internet security. It is widely used for network access control, VPN setup, and WEB applications. It supports a mutual authentication option in which both client and server hold X509 certificates and their associated RSA private keys. Sometimes, and according to the PKCS#11 and PKCS#15 standards, the user is equipped with a token storing a certificate and computing the RSA algorithm. This protocol is rather a single way authentication procedure, since a hacker may deliver a wrong certificate that will be probably approved by human user via a positive acknowledgement of the prompt notifying a security issue.

B. *Mutual authentication and EAP-TLS smart cards*

EAP-TLS is an IETF protocol (RFC 5216) which transparently shuttles TLS messages according to a datagram paradigm. In this context SSL sessions may be established without TCP/IP flavours. EAP-TLS smart cards were previously designed for access control issues in Wi-Fi networks [8]. A JAVA software (about 20KB), whose architecture was published in [9], runs in a tamper resistant device; its APDUs interface is described by an IETF draft [10] and performs an EAP-TLS stack both for client and server applications. It also manages a tiny certificate store that holds the authorized certification authority, the client (or server) certificate and the associated private key. The set of parameters required by the SSL stack is referred as the SSL-Identity.

There are multiple security benefits that are enforced by EAP-TLS smart cards. We claim a mutual strong authentication process because the server certificate is checked by a trusted computing environment and the client delivers a signature only to a trusted server. Furthermore both full and resume sessions are handled by the smart card; the master secret, which is the heart of TLS security is never exported from the secure micro-controller. In some cases the user mobility is enhanced because laptop is not configured with a set of X509 certificates; the SSL stack is physically transported by an USB dongle and is linked to a WEB browser via a plug and play mechanism.

C. *The TLS-Tandem Technology*

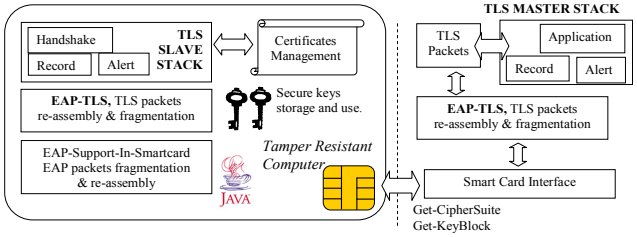

Figure 1. EAP-TLS smart card and the tandem concept

In many applications, TLS works in conjunction with the HTTP protocol. Schematically a HTTPS session requires two steps. First a pure TLS dialog performs the authentication between client and server, a set of ephemerals keys are computed that are used for the management of a secure channel enforcing information privacy and integrity. Second the previously opened secure channel transports HTTP messages that are encrypted and signed by a HMAC.

In the TLS-Tandem approach [11] the first TLS step is fully processed by the EAP-TLS smart card. When the authentication is completed ephemeral keys are exported from the smart card. Therefore two TLS stacks (see figure 1) are involved during an HTTPS session; first runs in the smart card and mainly manages handshake messages, second is instanced with the ephemeral keys (and associated cryptographic algorithms) exported from the secure device.

In the popular SSL implementations such as OpenSSL, a TLS session is managed thanks to three main APIs: *SSLOpen*, *SSLRead*, and *SSLWrite*. In the *tandem* context [11] the smart card fully performs the "*Open*" procedure, while "*Read*" and "*Write*" operations are executed by the host to which it is plugged.

Software glues needed for the interaction of TLS embedded stacks with WEB browsers are different depending of the host characteristics. We divide hosts in two categories, those who are not natively equipped with smart cards, for example PC and laptops, and those who are already incorporated a SIM card such as classical mobiles or smart phones.

D. *Deployment for laptops*

Smart cards are plugged to readers usually integrating an USB interface. Popular operating systems like Windows or Linux support specific APIs, named PC/SC (Personal Computer Smart Card), dedicated to readers discovery and use. Furthermore the JAVA language includes APIs compatible with PC/SC environment for version 1.6 and next ones. A proxy software [12] is a natural glue for starting HTTPS session from a WEB browser. It processes specific URL embedding a HTTPS request; it manages TCP/IP sockets, and forwards TLS packets to/from the smart card. When the TLS secure channel is established, it collects the TLS ephemeral keys and performs HTTP messages encryption and decryption.

Furthermore some USB devices integrate both smart card reader and FLASH memory (see figure 2). Thanks to these facilities, an EAP-TLS dongle may work in a plug and play way; the FLASH memory stores the proxy code, and at run time the proxy detects the smart card reader.

A proxy written in JAVA may be downloaded from a WEB server. Signed applets can be executed by a browser. This JAVA proxy is able to detect smart card readers. This attractive technology is still under evaluation.

Figure 2. A USB dongle including a javacard and FLASH (left part), and a 3G HSDPA modem including an USIM and SD memory card (right part)

E. *Deployment in mobile context*

Most of mobiles are equipped with SIM cards. It is estimated that about one billion of these devices embed a Java Virtual Machine. Our EAP-TLS is a java application compatible with this ecosystem.

There are basically two types of SIMs: SIM dedicated to GSM networks, and SIM designed for 3G/4G networks,

referred as USIM. For GSM networks, network facilities are managed by the smart card operating system. A single Javacard application doesn't conflict with GSM functionalities, but two JAVA applications can not usually run at the same time. For UMTS networks, network facilities are managed by a JAVA application. Hopefully USIM devices are able to manage at least two concurrent JAVA applications, and up to four may be supported according to a feature named "logical channel" introduced by the ISO7816 standard.

JAVA oriented operating systems, for example SYMBIAN, support APIs such as the JSR 177 ("Security and Trust Services API for J2ME^{TM}") able to exchange APDUs with SIM modules. *Middlet* software displays menus from which it is possible to start SSL connections and to download protected files.

Some smart phones or 3G USB dongles (see figure 2) support the AT-CSIM commands, which are an optional extension of classical AT facilities offered by most modems for multiple management purposes. AT-CSIM commands enable APDUs exchanges with SIM or USIM devices. For example we developed a proxy software working with commercial 3G dongles thank to AT-CSIM facilities and integrating a USIM device with two logical channels.

F. Computing Performances Issues

In this section we analyze computing performances issues, for two commercial devices, a Javacard and a USIM module, whose main characteristics are shown in figure 3.

Device	T_{MD5} /bloc (ms)	T_{SHA1} /bloc (ms)	$T_{RSA\ PUB}$ (ms)	$T_{RSA\ PRIV}$ (ms)	T_{3xDES} /bloc (ms)	T_{IO} (ms/ byte)
Java card	0,49	0,93	25	560	2,10	0,17
USIM	2,60	3,70	150	290	5,20	0,24

Figure 3. Basic Performances for commercial devices

According to the TLS protocol, two sessions modes are available: full (certificates are exchanged for the master secret calculation) and resume (a previous computed master secret is re-used). For 1024 bits RSA keys sizes, the following operations are realized:

- For the full mode, processing of 230 MD5 and 230 SHA1 blocs, one RSA encryption with a private key, one RSA decryption and one encryption with a public key. Furthermore 2500 bytes of information are exchanged.

- For the resume mode, processing of 75 MD5 and 75 SHA1 blocs; 250 bytes of information are exchanged.

The processing time for full session [8] may be written as

$$T_{full} = 230\ (T_{MD5}+T_{SHA1})+ T_{RSAPRIV} + 2\ T_{RSAPUB} + N\ T_{IO} + T_{SF}$$

And for the resume case [8]

$$T_{resume} = 75\ (T_{MD5}+T_{SHA1})+ N.\ T_{IO} + T_{SR}$$

where N is the number of information bytes exchanged, and T_{SF} and T_{SR} the residual time induced by embedded software instructions. We observe the following performances:

- For the Javacard, T_{full} = 2,0s T_{SF} = 0,7s - T_{resume}= 0,50s; T_{SR}= 0,35s
- For the USIM, T_{full} = 4,3s T_{SR}= 2,65s - T_{resume}=1,65s; T_{SR}= 1,45s

These results illustrate that commercial device manage SSL sessions in realistic times.

G. OpenID use case

OpenID [6] is an open Single Sign On framework dealing with three entities, the consumer a web site that requires an authentication, a user who wants to access to the service offered by the consumer, and the authentication server (AS). AS authenticates the user in a way that is not specified by the standard, for example a classical password may be used. An EAP-TLS smart card authenticates its bearer thanks an SSL session with mutual authentication (both server and client are equipped with an X509 certificate). Furthermore the embedded SSL stack enforces HTTP exchanges privacy and integrity. A demonstration of these functionalities was recently published in [13].

IV. IDENTITY MANAGEMENT MODEL

Considering the importance of security in SSO architectures, where identity usurpation can lead to dramatic consequences since the user is able to access a great deal of services with a unique authentication, a federated identity architecture aiming to be fully scalable must rely on security as its paramount key.

The architecture detailed in this section is a convergent identity management system, consisting of an identity management server based on EAP-TLS smart card authentication. The user is able to create and manage SSL identities that will be stored in its smart card, while the architecture itself can be integrated to any SSO framework in order to allow the user to easily access numerous service providers, hence offering a convergent, user-centric and secure solution to identity management.

A. Device Enrolment

The device enrolment corresponds to the initial phase of the identity lifecycle, which is illustrated in Figure 4, after the card manufacturing phase (step 1 of Figure 4). It consists in configuring a secure device embedding the TLS-Tandem technology, typically a smart card, which can itself be embedded in a 3G dongle, an USB storage dongle, or in a Smartphone as a USIM card. This card, which acts as an EAP-TLS client, will contain the user's SSL identities, as will be explained in section B. Since our architecture relies on PKI infrastructure and certificate issuance, one or several appropriate certificate authority (CA) must be defined that will sign the certificates issued by the Identity server, according to the trusted third party (TTP) principles of X509 infrastructure [15]. The CA can be for instance either the CA of the identity management server and/or the CA of the mobile phone company managing the (U)SIM card.

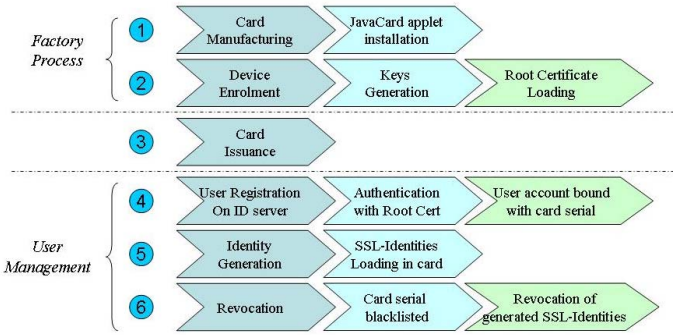

Figure 4. Card lifecycle from manufacturing to revocation

The configuration of the card in the device enrolment phase (step 2 of Figure 4), consists in providing it, before it is owned by the end user, with a unique serial number, the public key of the CA, and a pair of RSA public/private keys, from which a root certificate is issued after being signed by the CA. This root certificate will act as a first, impersonal identity stored on the card, thanks to which the user will be able to register to the identity management server. Another possible solution, consisting in issuing only a serial number with the card, was rejected, since there is no way for the user to register securely within a SSL session.

The root certificate subject shall be the card serial number, which is stored in the database storing the valid cards' serial number emitted. This choice will facilitate the user registration on the server, since the serial number of the card shall be checked in order to detect invalid cards.

Once the card has been issued to the end user (step 3 of the Figure 4), he can register to the identity server where his account will be created, making him able to personify his SSL-Identity card. Moreover, after the user has been registered, any exchange with the Identity server will be secured through an EAP-TLS session.

B. User Registration and Identity Generation

Once he has received his dongle, the user will need to activate it with the proper PIN code. Once the smart device has been activated, the user shall register a new account on the Identity Provider Website in order to associate his card's serial number to his Identity (step 4 of Figure 4). For this purpose, he will be asked to fill a form which shall be more or less exhaustive depending on the use case and the associated security level which will be expected. The information provided in this form will be requested later in case his dongle or Smartphone has been lost or stolen; for more details on this topic, see section D.

Upon registration of the user, his card serial number will be checked for validity, and added to the white list of serials stored within the Identity Provider database. The registration will also check the validity of the root certificate stored in the device, which implies that an EAP-TLS exchange with the identity server will be mandatory at this moment. Lastly, and after this step has been successfully achieved, the user can create an indefinite amount of Identities which will be connected to the card's serial number (step 5 of Figure 4). Before being stored in the smart card, the Identity will be created in the Identity server database, which makes it possible for the user to edit personal information related to his alias. Then, if enough space is left in the non volatile memory of the device, the user can create the corresponding SSL-Identity Container to be stored in the card. We think that the possibility of creating identities not necessarily stored in the card is useful, since the small capacity (about four Containers) of the non volatile memory of the card might be a constraint to the user.

The SSL-Identity Container mainly consists of a X509 certificate generated on demand by the user, the subject of which is the alias of the Identity. This certificate is signed by the CA, and encrypted as detailed in section C, before being loaded into the device.

Identities loaded in the smartcard are the one which the client will be able to activate while browsing the different service providers. Each time a client has a need of a specific Identity which is not loaded in his smartcard, the associated certificate can be generated and automatically stored in the card, granted that an SSL-Identity Container has been previously freed for this purpose.

C. SSL-Identity Encryption

Before being loaded into the card, an SSL-Identity Container will be ciphered by the Identity management server, following the scheme illustrated in Figure 5 and described hereafter. By adding a header to the Container, an SSL-Identity encryption key K followed by padding bytes is stored in the Container. Let K_{pub} and K_{priv} respectively design the public and private key of the card. The container C, before encryption, can be represented as in (1).

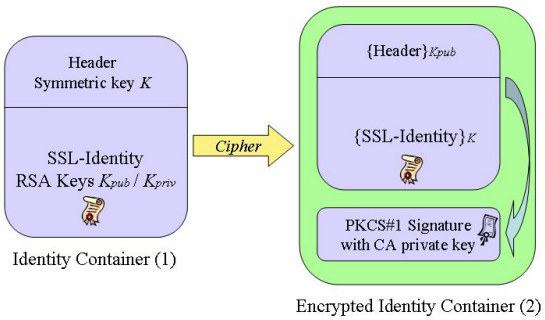

Figure 5. Formats of clear-text and encrypted Identity Containers

$$C = (K \mid Padding\ Bytes \mid Identity) \qquad (1)$$

The SSL-Identity Container C is encrypted, and the encrypted part is signed by the CA private key, according to scheme (2), resulting in an Encrypted Container EC.

$$EC = (\{K \mid Padding\ Bytes\}_{Kpub} \mid \{Identity\}_{K} \mid PKCS\#1\ Signature) \qquad (2)$$

Firstly, the card will check the PKCS#1 Signature since the public key of the CA is stored in its non volatile memory. Then, the card can decipher the header part with K_{priv} and then decrypt the Identity part with the symmetric key K it retrieved from the header. Lastly, it will check the validity of

the certificate contained in the Identity part, before storing it in its non volatile memory. Hence, any invalid container will be rejected, preventing fraudulent identity loading into the card.

D. Revocation

In case the user loses his dongle or his Smartphone, he must notify it so that his smart card's serial number is removed from the serials white list of the database and moved into a black list (step 6 of Figure 4). All the SSL-identities generated for this account and loaded into the smart device are revoked. In case the lost smartcard is trying to initiate a session with the Identity Provider, it will be physically destroyed, using an appropriate APDU coded for this purpose. Since a complete revocation of the user account seemed obviously troublesome for the user, we decided to adopt a more convenient solution in a user-centric context. Thus, in order to retrieve his Identity, the user will be sent a new dongle or Smartphone, he shall then make the proof of his Identity using the information he filled during step 4. The information required will be up to the Identity Provider: it can be based on a combination of information involving his card number, his phone number, an account number, an OTP given with the dongle, etc. If the information he submitted turns out to be valid, the serial number of the new smartcard he received will be added in the database and linked with his former account. The new SSL-Identity certificates generated and his new card will contain the new pair of private/public key associated to the serial number of the new card.

E. Performance Issues

The generation of SSL-Identity Containers is entirely performed by the identity server. The experimental platform uses *OpenSSL* in order to create the certificates. From the end user's point of view, the time needed for the generation of an Encrypted Container can be decomposed as in equation (3), where T_{CONT} is the time needed to generate a clear-text Container, T_{ENC} the time needed to encrypt the Container, and T_{SSL} the time needed to perform a SSL session with the server (using the root identity).

$$T_{generation} = T_{CONT} + T_{ENC} + T_{SSL} \quad (3)$$

Figure 6 shows measured times; containers were first generated and second encrypted by CGI scripts dealing with *OpenSSL*, and running in a WIN32 environment. The total $T_{generation}$ is deduced from equation (3).

T_{CONT}	T_{ENC}	T_{SSL}	$T_{generation}$
0,5s	0,3 s	2,5 s	3,2s

Figure 6. Measured time for each step of Container generation

Once the encrypted Container has been created, the user is able to download it into his smart device thanks to an AJAX interface described in [12]. It consists in sending about a dozen APDUs to the card, the half of it being a WRITE command in order to load the SSL-Identity Container, the other half being card management APDUs. The download time $T_{download}$ being measured is about 1,5 seconds, which can be decomposed as in equation (4), where n is the number of WRITE APDUs, T_{MGMT} the treatment time of the card management APDUs, and T_{DEC} the time needed by the card to decipher and check the validity of the SSL-Container.

$$T_{download} = n.T_{WRITE} + T_{MGMT} + T_{DEC} \quad (4)$$

The results of our measurements are displayed in Figure 7, with $n = 6$. These measurements illustrate the reality and the scalability of this identity management system.

T_{WRITE}	T_{MGMT}	T_{DEC}	$T_{download}$
75 ms	50 ms	1050 ms	1550 ms

Figure 7. Measured time for each step of Container generation

V. CONCLUSION

In this paper we have presented the foundations of a new convergent identity system based on the SSL-Identity concept. Technology basis is already validated for laptops and some 3G dongles. Our next work will aim at validating the scalability of this architecture.

REFERENCES

[1] Maler, E.; Reed, D., "The Venn of Identity: Options and Issues in Federated Identity Management" Security & Privacy, IEEE Volume: 6 , Issue: 2, 2008

[2] Microsoft, "Introducing Windows CardSpace", online, http://msdn.microsoft.com/en-us/library/aa480189.aspx

[3] Cantor, .S et al., "Shibboleth Architecture, Protocols and Profiles", 10 September 2005, online, http://shibboleth.internet2.edu/docs/internet2-mace-shibboleth-arch-protocols-200509.pdf

[4] Wason, T. et al., "Liberty ID-FF Architecture Overview", Liberty Alliance Project, online, http://www.projectliberty.org/resource_center/specifications/liberty_alliance_id_ff_1_2_specifications

[5] Cantor, .S et al., "Assertion and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard, 15 March 2005, online: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[6] OpenID community, "OpenID Authentication 2.0", online: http://openid.net/specs/openid-authentication-2_0.html

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

[8] 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

[9] Urien, P., Pujolle, G.,. "JavaCard for Emerging WLAN Environments". JavaOne Technical Session 2007 (TS -0285), JavaOne 2007, May 8-11, 2007, San Francisco, California, USA

[10] Urien, P., Pujolle, G., "EAP-Support in Smartcard", Internet Engineering Task Force Draft, draft-urien-eap-smartcard-18.txt, February 2010

[11] Urien, P., Elrharbi, S., "Tandem smart cards: enforcing trust for TLS-based network services", 8th International Workshop on Applications and Services in Wireless Networks (ASWN 2008), October 9th – 10th, Kassel, Germany

[12] Urien, P., "Collaboration of SSL smart cards within the WEB2 landscape", Collaborative Technologies and Systems, 2009. CTS'09. International Symposium on 18-22 May 2009 Page(s):187 – 194

[13] Urien, P., "An OpenID Provider based on SSL Smart Cards", Consumer Communications and Networking Conference, 2010. CCNC 2010. 7th IEEE 9-12 Jan. 2010

[14] Jorstad, I. et al., "Releasing the potential of OpenID and SIM", 13th International Conference on Intelligence in Next Generation Networks (ICIN): "Beyond the Bit Pipes", 2009

[15] Adams, C., Farrell, S., "Internet X.509 Public Key Infrastructure: Certificate Management Protocols", RFC 2510, March 1999.