# SECURE ACCESS MODULES FOR IDENTITY PROTECTION OVER THE EAP-TLS

## *Smartcard Benefits for User Anonymity in Wireless Infrastructures*

Pascal Urien, Mohamad Badra
*Networking and Computer Sciences Departement, ENST 37-39 rue Dareau, 75014, Paris France*
*Pascal.Urien@enst.fr, Mohamad.Badra@enst.fr*

Keywords: Identity Protection, EAP, PET, Security, Smartcard, TLS, WLAN.

Abstract: Identity protection and privacy became increasingly important in network communications; especially in wireless LAN. In this optic, Privacy Enhancing Technologies (PET) have been introduced to provide anonymous exchange and to protect personal data. In this paper, we present the SAM (Secure Access Module) architecture, which is a couple of smartcards (client and server) that process EAP-TLS, a transparent transport of TLS (Transport Layer Security) over EAP (Extensible Authentication Protocol). This architecture provides mutual authentication, identity protection and data un-traceability by preventing undesired and unnecessary processing of personal data.

# 1 INTRODUCTION

Without encryption features, information flowing in the network could be potentially logged, archived and searched. To resolve such problems, many security protocols, such as TLS (RFC 2246, 1999) and IPSec (RFC 2401, 1998) have been developed. Thanks to these protocols, communicating entities perform mutual authentication and compute a shared secret. This secret value is then used to generate cryptographic keys that enforce privacy and integrity services for information exchanged between these two parties.

Mutual authentication establishes the proof of identity and is usually realized thanks to X509 certificates or pre-shared-keys (PSK). Certificate deployment needs Public Key Infrastructures (PKI) and Certificate Authorities (CA) or Third Trusted Parties (TTP) to link the identity of the certificate owner to its public key. As for PSK, it requires neither of them and it is always identified through an ID value managed by the PSK owners. Consequently, PSK management does not require alike PKI infrastructures.

Even though security protocols are able to provide the basic security services, missing from their design is a way to protect information related to the communicating identities. Consequently, an intruder can easily learn who is reaching the network, when, and from where, and hence correlates client identity to connection location. For example, in the use case of TLS, the two communicating parties expose public-key certificates for mutual authentication and key establishment. However, TLS cannot prohibit an attacker to use traffic analysis to identify parties over time. In fact, during the TLS session certificates flow in clear through the network. Consequently, attacking TLS client's privacy becomes straightforward, because the intruder can easily read the client personal data, and make a correlation between its identity and its location, which is a privacy issue, especially to the wireless LAN (IEEE 802.11, 1999).

WLAN security relies on the IEEE 802.1x (IEEE 802.1x, 2001) infrastructure, which defines a framework for authenticating and controlling user's traffic, as well as dynamically exchanging encryption keys between the wireless terminal (supplicant) and the authentication server. It ties itself to a protocol called EAP, *Extensible Authentication Protocol* (RFC 3748, 2004) that is a powerful umbrella supporting multiple authentication methods and mechanisms, such as EAP-TLS (RFC 2716, 1999). However, security protocols lack the need to securely store secret and private keys. In fact, there is no safe place to store these critical credentials on disk, encrypted or

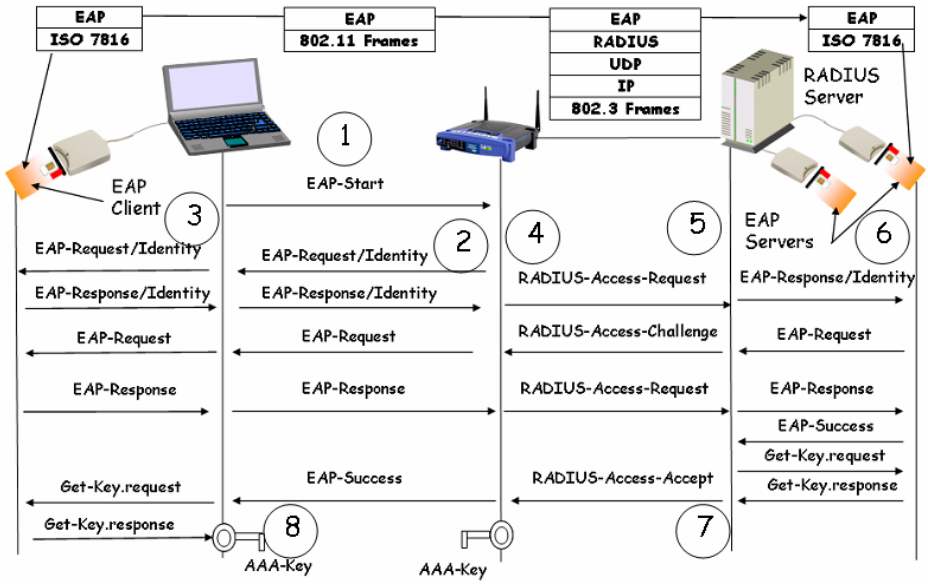


Figure 1: A SAM infrastructure for wireless LAN.

otherwise. The advanced approach for protecting these private data is to use tamper-resistance devices such as smartcards. These chips manage hardware and software countermeasures, which makes it difficult to extract or modify confidential information. In this way, we implemented EAP-TLS on smartcards.

In this paper, we introduce EAP-TLS servers, e.g. trusted applications running in Java cards (JavaCardForum, 2006) and processing EAP and TLS protocols. Furthermore, we define SAM (Secure Access Module) an architecture that enhances security authentication, identity protection, client credentials verification and services authorization. Next, we analysis this approach and we give an example of anonymous exchanges within TLS before concluding with some remarks.

## 2 INTRODUCING SAM IN WLAN

The extensible authentication protocol is a powerful umbrella that shelters multiple authentication scenari. It is the cornerstone of the IEEE 802.1x standard (IEEE 802.1x, 2001) which defines key exchange mechanisms between the wireless user (the supplicant) and the authentication server.

EAP messages are transported in 802.11 frames (IEEE 802.11, 1999),(IEEE 802.1x, 2001) between the supplicant and the access point, whereas RADIUS (RFC 3559, 2003) packets are used to convey these messages between the access point and authentication server. Between the terminal and the EAP smartcard, we defined a network interface that transports EAP messages using ISO 7816 payloads (ISO7816, 2006).

In (Urien *et al.*, 2004), we described an open architecture that processes EAP-TLS in smartcards and we presented a modular and flexible approach for controlling network accesses in WLAN environment. We also demonstrated that this approach is realistic with cheap smartcards, which are not specially designed for that purpose.

In this paper, we introduce EAP server smartcards that are dual forms of the EAP client smartcards. The main function of the EAP servers is to parse EAP responses, to process them and to return the appropriate EAP requests.

At the end of an EAP authentication session, the server notifies the success or the failure of the process. Upon success, it forwards the AAA key (see figure 1) to the access point; which is needed for all security operations that occurred between the access point and the client.

A dedicated RADIUS software processes RADIUS packets issued by the access point, extracts EAP messages, and forwards them to the appropriate EAP smartcard. We called SAM a couple of smartcards that processes EAP messages on both client and server side. Figure 1 describes the sequence of EAP exchanges between SAM during

the WLAN authentication phase. There are one EAP smartcard on each side, which parses and processes EAP messages.

In this sequence, the client intends to gain access to services provided by WLAN, and periodically transmits an EAP-Start (1) message to the access point to initiate an EAP authentication scenario. The access point produces, upon reception of this information, an EAP identity request (2), which will be forwarded to the client smartcard (3). This device will then deliver an appropriate EAP identity response, shuttled by ISO 7816 commands.

Upon reception by the client, the EAP identity response is encapsulated in a 802.11 frame (4), before conveying it to the access point that will in turn shuttle this response using a RADIUS packet (5).

SAM architecture uses many smartcards on the server side in order to process multiple authentication sessions. The server software checks the availability of one of its EAP-Server before forwarding (6) the EAP identity response to the available smartcard.

In reply, the EAP server sends an EAP request message, repeating the same encapsulation sequence before delivering the message to the client smartcard. This latter processes then the message and generates an appropriate response that will be sent to the selected EAP-Server.

Secret and cryptographic keys are always computed into the smartcards that also securely store network credentials (shared secrets, RSA private keys, certificates, etc.). Upon receiving the EAP-Success, the RADIUS (6) and the client (7) collect the AAA key from the SAM via a specific smartcard command. Thereafter, the client and the access point establish more exchange (for example 802.11i handshake) in order to compute cryptographic parameters that are required for radio security protocols, such as WEP and WPA.

# 3 EAP SERVER COMPONENTS

As we introduced, an EAP smartcard is a standardized, ISO 7816 microcontroller supporting most of authentication methods. It is described by an internet draft (Urien *et al.*, 2006b) and a more detailed description may be found in (Urien *et al.*, 2005).

We have developed the *OpenEapSmartcard* platform (OpenEapSmartcard, 2005) that provides a trusted execution environment for EAP methods and makes it a secure software entity. This platform is based on the standard Java Card Forum (JavaCardForum, 2006) framework that supports a cryptographic package including cryptographic resources needed by security protocols such as TLS.

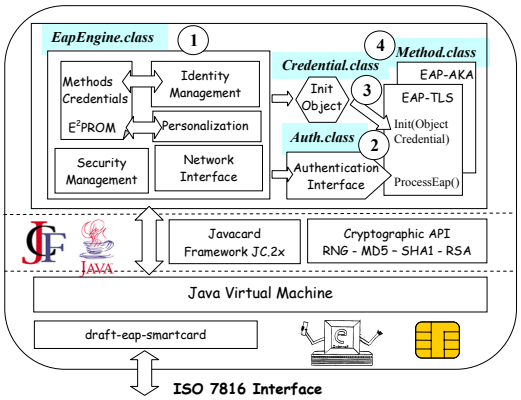


Figure 2: The OpenEapSmartcard architecture.

The *OpenEapSmartcard* platform comprises four java components (see figure 2):

1- The EapEngine: it manages several methods and multiple instances of a given method. It implements the EAP core and acts as a router that sends and receives packets to/from methods. At the end of an authentication session, each method computes a master cryptographic key (the AAA Key) which is collected by the terminal operating system.

2- The Authentication interface: this component defines all services that are mandatory in EAP methods in order to collaborate with the EapEngine. The two main functions are Init() and Process-Eap(). The first initializes method and returns an Authentication interface; the second processes incoming EAP packets. Methods may provide additional facilities dedicated to performances evaluations.

3- Credential Objects: each method is associated to a Credential Object that encapsulates all information required to process a given authentication scenario.

4- Methods: each authentication method is processed by a specific class. Once initialized, this object

analyses and processes each incoming EAP packet and delivers corresponding response.

# 4 BENCHMARK

## 4.1 Basic Constraints

The WLAN standard (IEEE 802.1x, 2001) specifies some timing constraint related to the delay between EAP requests and responses. That must be (by default) less than 30 seconds. Moreover, the duration of the authentication procedure is limited to a maximum value of 60 seconds, in order to avoid the client network interface reset, which is generated by the DHCP (RFC 2131, 1997) timeout.

In (Urien *et al.*, 2004) and (Urien *et al.*, 2006a), we described our implementation of EAP-TLS on EAP smartcards and detailed benchmark tests. The best results show that EAP-TLS session costs about five seconds on both on the client and server side. Even if the time required to run EAP-TLS is high – although it should be improved – it clearly appears that today javacards are able to fully run EAP-TLS in an interval compatible with IEEE 802.1X and DHCP requirements.

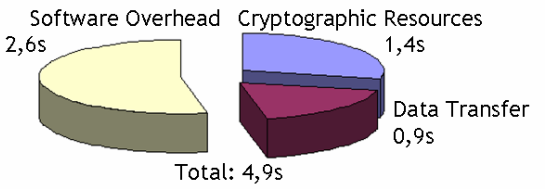


Figure 3: Computing times distribution for EAP-TLS smartcards.

Figure 3 shows the observed repartition of computing times for the EAP-TLS application. Operations are split into three categories, data transfer (about 2.6 kilobytes are exchanged during the TLS session with 1024 bits RSA keys), cryptographic resources (these facilities are supported by standard Javacard framework (Chen, 2000), and software overhead, which is the time required by all remaining operations.

## 4.2 EAP Server Card Performances

Due to modest performances of smartcards, each EAP packet is managed by a thread. A thread is started for each response and ends when the EAP-server delivers the next request or the last notification.

If we call *Tm* the main thread that runs the RADIUS server and *Ti* a thread associated to a given EAP message, the EAP server management is therefore done according to the following paradigm:

1- In thread Tm, the GetSession() procedure finds a smartcard (whose number is index) that is associated or that can be associated to the EAP session, identified with its id-session value, see (Urien *et al.*, 2006a) for more details. If no smartcard is available, then the incoming RADIUS Access-Request is silently discarded.

```
<Tm: index= GetSession(id_session)>
```

2- If a smartcard is available, then a thread Ti associated to that smartcard – identified by an index value – is created.

```
<Tm: StartThread(index)>
```

3- The incoming EAP-Response is forwarded to the appropriate smartcard (by ProcessEAP(index)), which afterwards returns an EAP request or a notification packet. Next, the thread Ti will build a RADIUS message (FormatRADIUSpacket()) that is sent to the access point (by SendRADIUSpacket()).

```
<Ti:ProcessEAP(index)
BuildRADIUSpacket() SendRADIUSpacket()>
```

As we previously mentioned, when no smartcard is available on the server side, the incoming RADIUS packet is silently discarded. This mechanism is similar to the classical blocking algorithm used in circuit-switching. Therefore, we shall estimate the system performance in a way similar to an M/M/C/C queuing system, from the Erlang-B formula.

$$p_c = \frac{(\lambda/\mu)^c}{c!}\left[\sum_{k=0}^{c}\frac{(\lambda/\mu)^k}{k!}\right]^{-1}$$

Where
- Pc is the probability of blocking (e.g. a RADIUS packet is silently discarded),
- c is the number of smartcards (EAP servers),
- $\lambda$ is the rate of authentication sessions, and
- $1/\mu$ the mean time of an authentication session.

With our best couple of devices (client and server), we observed that the measurement duration

of an authentication session is about 5+5 = 10 seconds; therefore $1/\mu = 10$. Let's now assume a network with 1000 users, authenticated every hour, then $\lambda = 1000/3600$ and so $\lambda/\mu = 10 \times 1000/3600 = 2{,}8$. As illustrated by figure 5, the probability of blocking (pc) is about 50% with 2 smartcards (c = 2) and only 1% with 8 smartcards (c = 8).

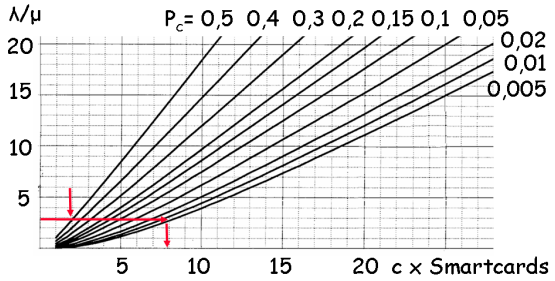


Figure 5: Using Erlang-B law, for adjustments of RADIUS server performances.

# 5 IDENTITY PROTECTION WITHIN TLS

As we introduced, we have implemented our SAM within Wi-Fi architecture, using EAP-TLS smartcards. EAP-TLS is based on TLS, the most deployed security protocol for securing exchanges. TLS provides connection security with peer entity authentication, data confidentiality and integrity, key generation and distribution, and security parameters negotiation.

TLS protocol consists of several sub-protocols (RFC 2246, 1999); specially the *Record* and the *Handshake* protocols. The *Record* protocol provides basic connection security for various higher layer protocols through encapsulation. The *Handshake* protocol is used to allow peers to agree upon security parameters for the Record layer, authenticate themselves, instantiate negotiated security parameters, and report error conditions to each other. Handshake results in security attribute negotiation. Once a transport connection is authenticated and a secret shared key is established with the TLS *Handshake* protocol, data exchanged by application protocols can be protected with cryptographic methods by the *Record* layer using the keying material derived from the shared secret.

We illustrate (see figure 6) TLS *Handshake* in three phases: the hello phase, the authentication phase, and the finished phase. During the hello phase, the client and the server negotiate cryptographic options (asymmetric and symmetric encryption algorithm, hash function, key exchange method, etc.) and exchange two random values that will be used by the key computation process. The second phase consists of exchanging certificates and of proving the identity and the validity of these certificates. In this phase, the client generates a secret called *PreMasterSecret*, which is sent encrypted using the server public key (see figure 6). During the finished phase, the client and the server exchange the *ChangeCipherSpec* and the *Finished* messages. The *ChangeCipherSpec* message is sent by both the client and the server to notify the receiving party that subsequent records will be protected under the newly negotiated cipher spec and keys. The *Finished* message is immediately sent after the *ChangeCipherSpec* message to verify that the key exchange and authentication processes were successful. The *Finished* message is the first message that is protected using the negotiated algorithms by the *Record* sub-protocol.

## 5.1 TLS Authentication Options

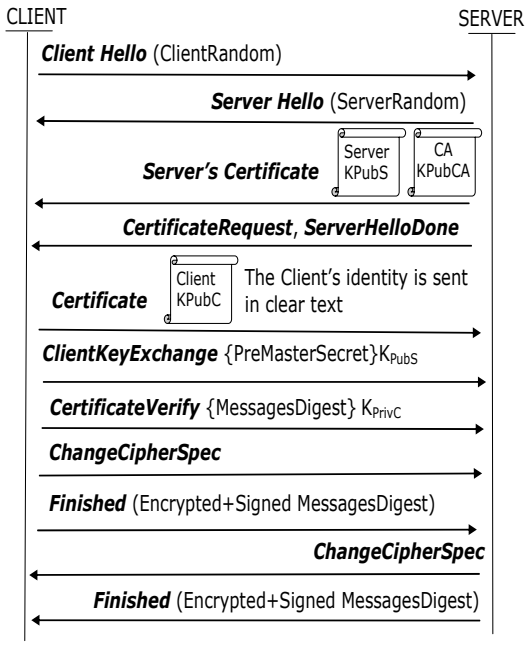


Figure 6: Mutual authentication with TLS, the client's identity is unprotected.

With TLS, the entities mutually authenticate using certificates which are sent in clear text,( see figure 6) leaving it unprotected. TLS supports three authentication modes: authentication of both parties,

server authentication with an unauthenticated client, and anonymity key exchange. For that, TLS proposes a range of cipher suites (cryptographic options). Some of these cipher suites provides anonymous communications in which neither party is authenticated. However, anonymous cipher suites are strongly discouraged because they cannot prevent man-in-the-middle attacks.

## 5.2 TLS Identity Protection

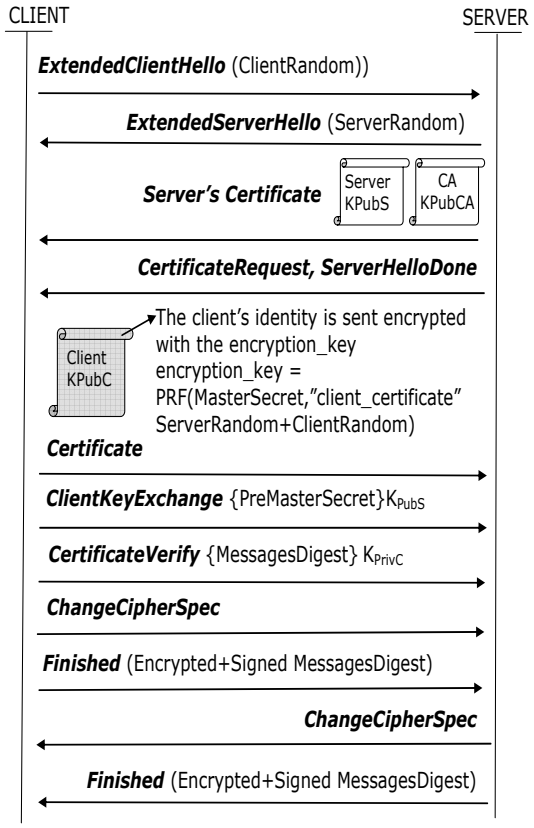


Figure 7: TLS Handshake with identity protection.

There are some propositions to allow identity protection with TLS. One of them establishes an initial anonymous Diffie-Hellman exchange before establishing an ordinary handshake with identity information (Rescorla, 2000), even though this wouldn't be secure against active attackers. And it wouldn't favorable for some environments (e.g. mobile) because the client and the server must encrypt the whole ordinary TLS session and then increase enormously the processing time to establish the secure session. Another solution is a matter of changing the order of the messages that the client sends to the server in order to activate the encryption/decryption before sending the certificate. That way the certificate is sent protected by the new bulk encryption algorithm and key. However, this solution requires the definition of new version for TLS, in which IETF TLS working group does not agree (Rescorla, 2000). In order to provide identity protection in an extensible way and to integrate that with the SAM architecture, we extend TLS with a new extension using the TLS extensions standard (RFC 3546, 2003). This latter describes ways to add functionality to TLS (RFC 2246, 1999). The standard provides generic extension mechanisms for the TLS handshake client and server hellos and specifies some extensions using these mechanisms. It specifies extensions using the following generic mechanism represented in the external data representation (XDR) format (RFC 1832, 1995).

```
struct {
ExtensionType extension_type;
opaque extension_data <0 .. 2^16-1>;
       } Extension;
```

The extension_data field contains information specific to the particular extension type (identified in the extension_type field). The extension defined below is sent by the client to indicate to the server that the client certificate will be sent encrypted using the negotiated symmetric algorithm and a secret key. It contains the symmetric encryption algorithms supported by the client in order of the client's preference (favorite choice first). The defined extension is of type "identity_protection". The "extension_data" field of this extension shall contain:

```
struct {
SymmetricAlgorithm
symmetric_algorithm_list<0..2^16-1>;
       } IdentityProtection;
enum {rc4(0),(255)} SymmetricAlgorithm;
```

If the server is not able to negotiate such session, it replays with an alert notification, falls back on an ordinary TLS handshake or stops the negotiation.

If the server is able to process this extension, it selects a symmetric encryption algorithm from the list sent by the client. The selected algorithm and a 16-bytes encryption key will be then used by the client to encrypt its certificate (in mode stream). Both the client and the server compute the encryption key by applying the PRF-TLS function (RFC 2246, 1999) on the random values and the master secret.

```
encryption_key =
PRF(master_secret,
"client_certificate",
ServerHello.random+ClientHello.random);
```

Upon receipt of the encrypted client certificate, the server should decrypt it, and check that the certificate is valid. Next, the client and the server continue their exchange as defined in TLS.

Figure 7 illustrates TLS handshake, with our proposed identity protection mechanism. Identity protection is negotiated through two TLS extensions included in client and server *Hello* messages. The client's certificate, which is usually sent in a clear text, is encrypted according to the negotiated cryptographic algorithm associated to the *encryption_key*, defined above. More details have been published in an IETF draft (Urien *et al.*, 2006c).

## 5.3 Identity Protection with SAM

As we cited before, it's desirable to manage security protocols parameters, such as private and secret keys, by tamper-resistance computers. In this optic, our SAM smartcard allows highly secure storage of such credentials and verifies certificates in a trusted environment. It is, moreover, the only entity in the chain that retrieves certificates in clear text. In other words, all TLS cryptographic computations and certificate encryption/decryption are performed into the SAM. In this way, the certificate will not flow unencrypted nor on the network, neither on the client or server machines.

# 6 CONCLUSION

Identity protection is a critical requirement for network's users, especially in a wireless context. In this paper, we introduced the SAM concept that works in WLAN or VPN architectures. Next, we extended the TLS protocol to provide identity protection services, and we integrated it within SAM infrastructures. The use of smart cards allows trusted computing, ensures client identity protection, and guaranties safe storage of sensitive credentials.

# REFERENCES


RFC 1832, 1995. XDR: External Data Representation Standard. *Internet Engineering Task Force, IETF*.

RFC 2131, 1997. Dynamic Host Configuration Protocol, DHCP. *Internet Engineering Task Force, IETF*.

RFC 2401, 1998. Security Architecture for the Internet Protocol. *Internet Engineering Task Force, IETF*.

IEEE 802.11, 1999. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, *Institute of Electrical and Electronics Engineers*.

RFC 2716, 1999. PPP EAP TLS Authentication Protocol. *Internet Engineering Task Force, IETF*.

RFC 2246, 1999. The TLS Protocol Version 1.0. *Internet Engineering Task Force, IETF*.

Chen, C., 2000, *Java Card Technology for Smart Cards*. The Java Series, Addison Wesley, 2000.

Rescorla, E., 2000. *SSL and TLS- Designing and Building Secure Systems*, Addison Wesley, 2000.

IEEE 802.1X, 2001. "Local and Metropolitan Area Networks: Port-Based Network Access Control", *Institute of Electrical and Electronics Engineers*.

RFC 3546, 2003. Transport Layer Security (TLS) Extensions. *Internet Engineering Task Force, IETF*.

RFC 3559, 2003. Remote Authentication Dial In User Service Support for EAP. *Internet Engineering Task Force, IETF*.

RFC 3748, 2004. Extensible Authentication Protocol, (EAP). *Internet Engineering Task Force, IETF*.

Urien P., Badra M., and Dandjinou M., 2004. EAP-TLS smartcards, from dream to reality. In ASWN 2004, *Fourth workshop on Applications and Services in Wireless Networks*,. Boston, USA.

OpenEapSmartcard, 2005. WEB site, http://www.enst.fr/~urien/openeapsmartcard.

Urien P., Dandjinou M., 2005. The OpenEapSmartcard project. Short paper, In ACNS 2005, *Applied Cryptography and Network Security 2005*, Columbia University, New York, USA

ISO 7816, 2006. Identification cards-Integrated circuit(s) card with contact, *International Organization for Standardization (ISO)*, ISO/IEC 7816.

JavaCardForum, 2006. www.javacardforum.org

Urien P., Dandjinou M., 2006a. Introducing Smartcard Enabled RADIUS Server, In CTS 2006, *the 2006 International Symposium on Collaborative Technologies and Systems*, Las Vegas, USA.

Urien P., Pujolle, G., 2006b. EAP support in smartcard. Internet Draft, *Internet Engineering Task Force, IETF*.

Urien P., Badra M., 2006c. Identity Protection within EAP-TLS, Internet Draft, *Internet Engineering Task Force, IETF*.