The OpenEapSmartcard platform

Pascal Urien[1], Mesmin Dandjinou[2]

[1]ENST 36/38 rue Dareau 75014 Paris France,
[2]Université Polytechnique de Bobo-Dioulasso, Burkina Faso
Pascal.Urien@enst.fr, Mesmin.Dandjinou@voila.fr

Abstract. This paper presents the first javacard platform dedicated to IP (Wireless) LAN security issues. We have defined an open architecture that processes Extensible Authentication Protocol (*EAP*) in smartcards, which is the standard defined by IETF[1] and IEEE-802[2] committees for users' authentication in various network environments like *Wi-Fi, WiMax,* or *IPSEC*.[3] These tamper resistant devices are generally considered as the most trusted computing platforms. They have been selected by the DoD[4] for military ID cards, by the Belgium government for citizen ID cards, and they will be included in US and European passports. Although secure, javacards are cheap and manufactured by many companies. We present and analyze results obtained with five different smartcards, for two authentication scenari. The first works with an asymmetric algorithm (EAP-TLS, a transparent transport of the well known SSL[5] standard), the second uses a pre-share key scheme (EAP-PSK) based on the AES algorithm and the One-Key CBC MAC function (OMAC), which is under consideration by NIST[6] for standardization. We demonstrate that this open and flexible approach, is working with existing components, although performances enhancement is necessary.

KEYWORDS: Security, WLAN, smartcards, javacards

1 Introduction

Wireless IP networks, e.g. cheaper 802.11 technologies, follow an exponential growth. Everyday more and more people use wireless IP services at home, at office or in the city. However security issues still remain the Achilles' heel of these emerging ubiquitous networks [10], [11]. Even when Wi-Fi accesses are free, wireless clients need privacy, and organizations providing wireless infrastructures must control and manage network accesses.

[1] Internet Engineering Task Force.
[2] Institute of Electrical and Electronics Engineers, IEEE 802 LAN/MAN Standards Committee.
[3] IP Security Protocol.
[4] United State Department Of Defense.
[5] Secure Sockets Layer.
[6] National Institute of Standards and Technology.

Please use the following format when citing this chapter:

Urien, P. and Dandjinou, M., 2007, in IFIP International Federation for Information Processing, Volume 229, Network Control and Engineering for QoS, Security, and Mobility, IV, ed. Gaïti, D., (Boston: Springer), pp. 75–86.

After five years of efforts, standardization committees have built a secure architecture based on standards like WEP [8] (Wireless Equivalent Privacy), IEEE 802.1x [12], IEEE 802.11i [18] and Extensible Authentication Protocol [19]. The network user (*Supplicant*) is authenticated (via the EAP protocol) by a remote (*RADIUS*) server. Upon success of this operation, a master key is computed by these two entities, from which are deduced all parameters required by radio security protocols (WEP, TKIP, 802.11i), in order to provide privacy (encryption) and data integrity services.

This paper presents the first open architecture for processing the EAP protocol in JAVA smartcards (supplicant side). The basic idea is to define a framework suitable for organizations that intend to independently manage their wireless network security, according to symmetric (shared secret) or asymmetric (RSA keys and certificates) infrastructures.

Section one discusses about smartcard benefits for wireless security management. Section two presents basic technical constraints and the *OpenEapSmartcard* platform. Section three analyses experimental results obtained for EAP-TLS method [6] (SSL based authentication method) with various smartcards. Section four comments results observed for EAP-PSK method [22], based on the AES algorithm [12] that has been recently proposed as RFC at the IETF committee.

1.1 Smartcard benefits for wireless network security management

Reliable and cheap 802.11 technologies make it possible for companies, administrations, or cities to deploy wireless networks supporting data (WEB, email) or multimedia (voice, images) services. There is a need to control network accesses for confidentiality purposes (if wireless resources are restricted to authorized staff) or for legal issues (if some behaviors are forbidden in open networks).

The basic choice for user's authentication is using password or not. A good password is difficult to memorize, although it can be easily duplicated or stolen.

Smartcard is an alternative to passwords. It's a secure and cheap silicon slice [26], whose area is about 25 mm^2. It includes a CPU (8 to 50 MHz clock), ROM (up to 256 KB), non volatile memory (128 KB of E^2PROM or up to one MB of FLASH) and RAM (from 4 to 8 KB). Information is exchange via a serial link, whose throughput ranges between 9600 and 230,000 bit/s.

About one billion of SIM smartcards are manufactured each year[7]. SIM module [4] manages subscriber' authentication in GSM networks; it stores its

[7] Source, www.eurosmart.com

identity (IMSI) and computes a symmetric algorithm (A3/A8) associated to a secret key (Ki). Almost every SIM embeds a Java Virtual Machine (JVM); according to its price, a crypto processor supports additional cryptographic facilities like RSA or triple DES. Because it's possible to download applets in such components [5], which may process complex cryptographic protocols, we believe that they are very well adapted for wireless security issues.

Smartcards are generally considered as the most secure computing environment [23]. They have been selected by the DoD for military ID cards [24], by the Belgium government for citizen ID cards [25], and they will be included in US and European passports. Hundred of millions people used cash cards including tamper resistant chips running payment standards like BO' or EMV. Generally smartcard is unblocked by a *Personal Identification Number* (PIN) code whose size is 4 digits; the system is frozen after three wrong tries. Furthermore it is already possible to get components working with fingerprint recognition [13].

2 OpenEapSmartcard

Contrary to mobile phone operators that manage worldwide networks, we believe that IP wireless infrastructures could look like a constellation of "small" domains hold by public authorities (cities, campus...), private companies or individuals. It's likely than each of them will control network accesses, according to specific mechanisms and policies. As an illustration more than fifty EAP type, e.g. different authentication methods are already registered by the IANA[8].

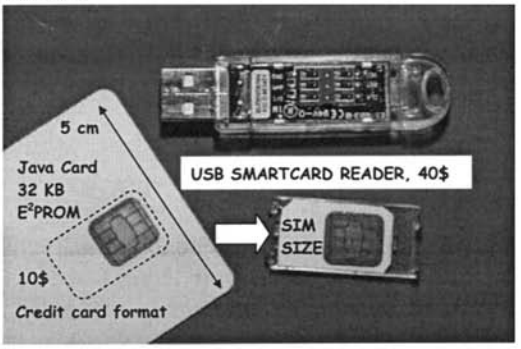

Figure 1: illustration of smartcard form factors and prices

The first requirement for *OpenEapSmartcard* initiative is ***flexibility***. In the past many security threats were discovered for GSM (COMP 128-1 crack) or

[8] Internet Assigned Numbers Authority.

802.11 (WEP crack). Open code is a good security principle that enables code reviewing; it facilitates study of multiple security architectures; it allows evaluation of deployment costs induced by smartcards price or infrastructure management (secret key management, certificates management, communication reliability,...).

The second requirement is ***feasibility***. In EAP context an authentication is a suite of requests sent to supplicant (the smartcard in our proposal) that produces responses. As specified in [12] the default return time trip (RTT) must be less than 30s. Another constraint is the complexity of authentication protocols; code byte is limited by non volatile memory capacities that currently range between 32 and 128 KBytes; hopefully this size could quickly evolve toward the MByte

The last requirement is **low costs** and **multi form factors**. Figure 1 illustrates this point, Javacard costs less than 10$ per unit, and USB readers are available for less than 40$.

As we will show it later, complex methods like EAP-TLS [6] or EAP-PSK [22] need about 20 KB of E^2PROM, and may work with RTT less than 30s with an authentication time comprised between 10s and 45s.

2.1 About java card platforms

They are two kinds of Javacard platforms.

First class, that we refer as general purpose devices, implements APIs [9] defined by the Javacard forum[9] whose current version are JC2.1 and JC2.2. All these releases support cryptographic facilities like RSA, MD5, SHA1 and random number generators (RNG). AES algorithm is only available beyond version 2.2.

Second class is based on SIM chips, and is described by the TS 03.19 standard [5]. It supports JC2.x classes and additional APIs that access to files embedded in these components.

From a functional point of view these two environments are quite similar. Although the GSM platform is widely available and cheap, it doesn't usually embed asymmetric cryptographic facilities that are not used in GSM networks.

2.2 Software architecture

The software architecture mainly comprises four java components

- The *EapEngine* which implements the *EAP core*, and acts as a router that sends and receives packets to/from authentication methods

[9]www.javacardforum.org

- An *Authentication* interface that defines all services offered by EAP methods
- A *Credential* object which stores information needed for method initialization.
- One or more *Methods* that instantiate authentication scenari like EAP-TLS or EAP-PSK.

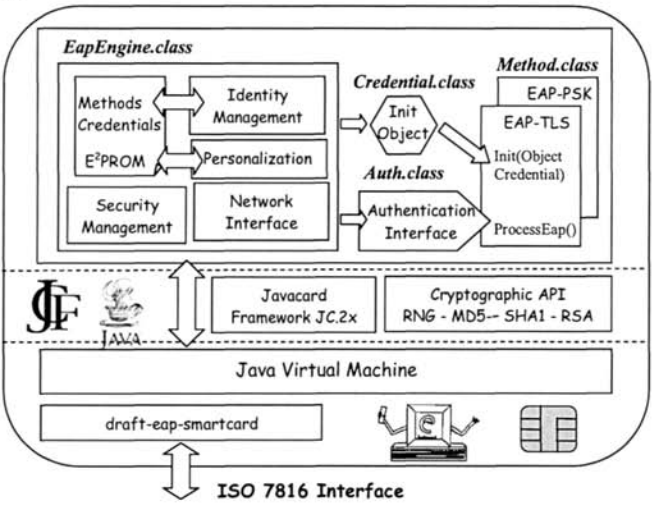

Figure 2. OpenEapSmartcard software architecture

2.2.1 EapEngine

Four services are offered by this module

- *Network interface.* Incoming EAP requests are checked and forwarded to the appropriate method. At the end of authentication process, each method computes a master cryptographic key (PMK) which is read by the supplicant operating system.
- *Identity management.* Smartcard manages several methods and/or multiple instances of the same one. An identities list stores credentials (EAP-ID, X509 certificates, RSA keys, shared secrets) required by embedded methods. This service allows to browse available identities and to select one of them.
- *Security management.* Smart card is protected by two PIN codes, one for its issuer and the other for its holder. In future versions this service could manage fingerprint recognition.
- *Personalization management.* Identity items and PIN codes are controlled and set by smartcard issuers. Some authentication methods like EAP-TLS or EAP-PSK create a secure channel. This protected link may be used for remote

management, which could modify *"over the air"* parameters embedded in tamper resistant chip. This model is closed to standard TS 03.48, which is widely deployed in GSM network for SIM card update purposes.

The ISO 7816 standard [1] defines logical structures of commands (*APDUs*) and responses exchanged with smartcards. APDUs understood by the *EapEngine* are described and explained by an internet draft [21].

2.2.2 Authentication Interface

This component defines all services (see figure 3) that are mandatory in EAP methods, in order to collaborate with *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 (fct()) dedicated to performances evaluations.

Interface auth	
void	`fct(javacard.framework.APDU apdu, byte[] in, short inlength)` Method functions apdu: incoming APDU in: buffer associated to the incoming APDU inlength: P3 value
byte[]	`Get Fct Buffer()` Returns a function buffer
short	`Get Fct Length()` Returns a function buffer length
short	`Get Fct Offset()` Returns a function buffer offset
byte[]	`Get Out Buffer()` Returns the response buffer
short	`Get Out Length()` Returns the response buffer length
short	`Get Out Offset()` Returns the response buffer offset
auth	`Init(java.lang.Object credentials)` Method Initialization
boolean	`IsFragmented()` Fragmentation in progress
boolean	`IsLongFct()` Indicates that the response of a function is stored in a private buffer
boolean	`IsLongResponse()` Indicates that the response of the method is stored in a private buffer
short	`process eap(byte[] in, short inlength)` Method Processing in: incoming APDU buffer inlength: length of the incoming APDU Returns -length of the response -negative value if an error occurred
void	`reset()` Resets the method
short	`status()` Gets the method status

Figure 3. Authentication Interface

2.2.3 Credential object

Each method is associated to a *Credential* Object (see figure 3, *Init()*) that encapsulates all information required for processing a given authentication scenario.

2.2.4 Method

Each authentication scenario is processed by a specific Method class. Once initialized, it analyses each incoming EAP request and delivers corresponding response. The number of embedded methods is limited by the smartcard non volatile memory (E^2PROM) size.

2.3 Integration to terminals

Some operating systems, like win32, already support the EAP protocol for wired and wireless networks. A given EAP authentication scenario is processed by a particular dynamic library (DLL) named EAP-Provider [17]. A *generic* library that uses PC/SC [2] services (plug and play support for smartcard readers), forwards incoming EAP requests to EAP smartcard that computes corresponding responses.

3 EAP-TLS smartcard

The EAP-TLS authentication scenario [6] works with the IETF version of the well known SSL protocol. Additional cryptographic features that are not supported in JC2.x framework are written in java, like HMAC [3], RC4, TLS PRF ([7] *Pseudo Random Function*) and X509 certificate parser.

We have implemented two modes for EAP-TLS. The first is asymmetric and works with a mutual authentication based on RSA keys and X509 certificates. The second called session resume mode [7], is symmetric and is based on Session-ID and Master-Secret parameters computed during a previous (full and asymmetric) session. This last mode may be useful for SIM javacards that don't include RSA crypto-processor.

3.1 Experimental results

Tests were performed on four smartcards, equipped with 8 bits processors (8 MHz clock), issued by different manufacturers. Devices A, C and D are general purpose javacards, component B is a SIM module. Two of them that have RTT times less than 30s and may be used for authentication in existing 802.11 wireless infrastructures.

3.1.1 TLS "full mode"

During the authentication scenario about 2,500 bytes are sent and received. The time consumed by this operation ranges between 2 and 5 seconds. This parameter is dependant on three constraints; physical throughput,

performances of embedded JVM, and time required for writing operations in E^2PROM.

RSA resources, provided by crypto-processor, are used three times. Public key operations need around 200ms, and encryption with private key is rather fast, and consumes less than 500 ms.

According to the TLS specification, three dual digests (MD5 + SHA1) are performed on these exchanged data, e.g. about 7500 bytes (120 blocs of 512 bits). We observe (see table1, Dual Hash column) average computing times (per bloc) ranging between 3,7 ms (900ms/240) and 24 ms (5700ms/240)

Five occurrences of pseudo random function (PRF), imply the calculation of 31 HMAC-MD5 and 31 HMAC-SHA1 procedures [3], which process 140 MD5 blocs and 140 SHA1 blocs. Each HMAC computes 2 digests, whose average size is about 2, 25 blocs. We notice (table 1) an important overhead, induced by our java implementation, excepted for the device D.

RC4 algorithm is fully written in java, observed performances clearly illustrate various behaviors of multiple embedded virtual machines.

Device	RTT MAX	Total Authentication Time	Data Transfer 2500 bytes	Dual Hash 2x120 blocs	PRF 2x140 blocs	RC4 2 x 32 bytes	OTHER
A	52,3s	78,1s	2,3s 2,9%	5,7s 7,3%	42,4s 54,2%	14,7s 18,9%	13,0s 16,6%
B	21,0s	34,3s	4,3s 12,5%	4,5s 13,2%	19,7s 57,3%	2,3s 6,7%	3,5s 10,2%
C	22,3s	33,3s	4,9s 14,7%	3,5s 10,4%	13,3s 40,0%	2,8s 8,4%	8,8s 26,5%
D	5,2s	9,3s	1,6s 17,2%	0,9s 9,7%	4,5s 48,4%	0,7s 7,5%	1,6s 17,2%

Table 1. Measured authentication time for four smartcards, TLS full mode

Device	RSA 1024 bits Public Key Initialization	RSA 1024 bits Encryption	RSA(Verify Data) 1024 bits Private Key Encryption
A	0,21s	0,16s	0,33s
B	-	-	< 0,80s
C	0,31s	0,38s	0,43s
D	-	0,02s	0,11s

Table 2. RSA performances for four smartcards

3.2 Resume mode

In this mode around 250 bytes are exchanged, a previously computed master-secret is re-used, and no RSA resources are necessary. Two dual digests (MD5 + SHA1) are performed on about 2x150 bytes (6 blocs of 512 bits)

Four occurrences of the PRF functions imply the calculation of 15 HMAC-MD5 and 15 HMAC-SHA1 procedures which process 108 MD5 blocs and 108 SHA1 blocs. Each HMAC computes 2 digests, whose average size is about 3,6 blocs.

In summary (see table 3) the resume mode is quicker than the normal one, it may be useful for fast user's re-authentication.

Device	Total Authentication Time	Data Transfer 230 bytes	Dual Hash 2x6 blocs	PRF 2x108 blocs	RC4 2x32 bytes	OTHER
A	49,5s	0,9s 1,9%	0,3s 0,6%	32,5s 65,6%	14,7s 29,7%	1,1s 2,2%
B	18,7s	1,0s 5,4%	0,2s 1,0%	15,2s 81,3%	2,3s 12,3%	0,0s 0%
D	5,5s	0,2s 3,6%	0,0s 0%	3,5s 63,7	0,7s 12,7%	1,1s 20,0%

Table 3. Measured authentication time for three smartcards, TLS resume mode

3.3 Performances limits

	Full Mode					Session Resume Mode			
	Total Time	Data Transfer (2500 bytes)	Dual Hash 2x120 blocs	PRF 2x140 blocs	RSA	Total Time	Data Transfer (250 bytes)	Dual Hash 2x 6 blocs	PRF 2x108 blocs
A	15,6	2,3	5,7	6,7	<0,9	6,4	<1,0	0,3	5,1
B	16,5	4,3	4,5	5,3	<2,4	5,4	<1,0	0,3	4,1
C	14,0	4,9	3,5	4,1	<1,5	4,0	<1,0	0,2	2,8
D	3,7s	1,6	0,9	1,0	<0,2	1,0	<0,2	0,0	0,8

Table 4. TLS computing time estimation, obtained by neglecting java overhead, for PRF and RC4 calculation

Table 4 estimates ultimate authentication times for full and resume modes, by neglecting the java overhead. Scenario duration is only dependent on data transfer, number of digest operations, and RSA calculations. It clearly appears that performances are strongly linked to digest computing; although IO throughput can't be neglected for full mode operation.

4 The EAP-PSK smartcard.

The EAP-PSK method [22] is a symmetric authentication scenario, recently proposed at the IETF committee, and based on AES algorithm. We are going to briefly analyze its behavior. Server and client use a (secret) pre share key (PSK, 128 bits), from which are deduced two AES (128 bits) keys: AK (Authentication Key) and KDK (Key-Derivation Key).

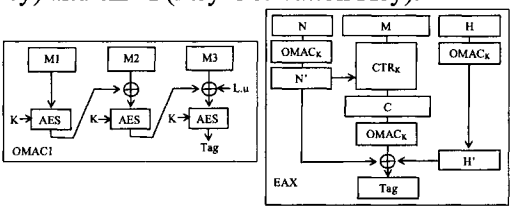

Figure 4: OMAC1 (left part) and EAX (right part) principles; K represents an AES key.

These two entities exchange random values (RAND_P, RAND_S) which are associated to MAC (128 bits) values (MAC_P, MAC_S) computed according to the OMAC [15] algorithm, which is under consideration by NIST for standardization. This algorithm splits incoming messages in blocs (128 bits) encrypted by AK key and mixed with EXOR operations (see figure 4).

A *Modified Counter Mode* [14] (MCM) algorithm computes from the RAND_P field and the KDK key, the *Transient EAP Key* (TEK) and the IEEE 802.1x *Pairwise Master Key* (PMK).

EAX [20] is a block-cipher mode of operation, for solving the problem of authenticated encryption with associated header. It works with a nonce N, a message M, and a header H. It is used by EAP-PSK to provide a secure channel (S_CHANNEL), protected by the TEK key that transports at least, an encrypted and tagged *status byte*. EAX required three OMAC instances (for the nonce, the header and tag, see figure 4) and a counter mode encryption (*CTR* mode) for message ciphering.

Device	AES SET KEY	AES CIPHER (T_{AES})	OMAC 25 bytes 3xAES $T=n_{AES}(T_{AES}+p)$	OMAC 49 bytes 5xAES $T=n_{AES}(T_{AES}+p)$	AEX (N=16 bytes, H=5 bytes, M=1 byte) 10xAES
E	0,018s	0,011s	0,114s	0,170s	1,022s
C	2,6s	0,39s	1,30s	2,24s	5,13s

Table 5. Basic parameters for EAP-PSK

Most of today available Javacards implement the JC2.1 standard, which doesn't support the AES algorithm, but hopefully it is included in JC2.2 smartcards. Consequently our tests deal with two kinds of AES instances, full

Java (device C) or native implementation (device E). As illustrated by table 5 the pure software version is rather slow (0,4s per block encryption), specially when it's necessary to initialize a key (1.3s). Additional cryptographic resources (AEX, OMAC, ...) are provided by java classes.

		Device E		Device C	
Operation	Number of AES Encrypted blocs (n)	Computing time CT	Estimated Java overhead CT-n.T_{AES}	Computing time CT	Estimated Java overhead CT-n.T_{AES}
First Request					
SET_KEY(AK)		0,018		2,60	
OMAC (49 bytes)	5	0,170	0,110	2,24	0,29
Second Request					
OMAC (25 bytes)	3	0,114	0,079	1,30	0,13
SET-KEY(KDK)		0,018		2,60	
MCM	6	0,070		2,34	
SET-KEY(TEK)		0,018		2,60	
AEX(N=16,H=5,M=1)	10	1,022	0,906	5,13	1,23
AEX(N=16,H=5,M=1)	10	1,022	0,906	5,13	1,23
TOTAL	34	2,45s	2,00s	23,90s	2,90s

Table 6. EAP-PSK computing time

We (not surprisingly) observe (see table 5) that computing time of OMAC increases proportionally to the number of (AES) blocs. The (Java) penalty per bloc (p, table 5) is about 20ms for device E and 50ms for device C. The EAX procedure is more complex and induces further Java additional computing times whose values are 0,9s for device D and 1,2s for device B. We finally notice for device E, that total computing time (2,45s) is mainly due to Java overhead because the AES penalty (34 x 0,011) is not the predominant factor.

Table 6 details observed computing times, and clearly illustrates that EAP-PSK is faster than EAP-TLS resume mode.

5 Conclusion

In this paper we have presented an open, modular and flexible approach for controlling network accesses in WLAN environment. We have demonstrated that this proposal is realistic, even with cheap smartcards that are not specially designed for that purpose. We have published [27] these source codes and we plan to develop an open library that will cover more scenari.

6 References

[1] International Organization for Standardization (ISO) "Identification cards - Integrated circuit(s) card with contact" ISO/IEC 7816.

[2] PC/SC (1996), Interoperability Specification for ICCs and Personal Computer Systems, © 1996 CP8 Transac, HP, Microsoft, Schlumberger, Siemens Nixdorf.
[3] H. Krawczyk, M. Bellare, R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, September 1997.
[4] ETSI - GSM 11.11 "Digital cellular telecommunications system (Phase2+); Specification of the Subscriber Interface Identity Module – Mobile Equipment (SIM_ME) interface".
[5] ETSI GSM 11.19, "Digital cellular telecommunications system (Phase 2+); GSM API for SIM toolkit stage 2"
[6] B. Aboba, D. Simon, "PPP EAP TLS Authentication Protocol", RFC 2716, October 1999.
[7] T. Dierks, C. Allen, , "The TLS Protocol Version 1.0", RFC 2246, January 1999
[8] Institute of Electrical and Electronics Engineers, "Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Standard 802.11, 1999.
[9] Zhiqun Chen, "Java Card Technology for Smart Cards: Architecture and Programmer's Guide", SUN book, 2000
[10] N. Borisov, I.GoldBerg, D.Wagner, Intercepting Mobile Communications: The Insecurity of 802.11, Proceeding of the Eleventh Annual International Conference on Mobile Computing And Network, p180, July 16-21, 2001.
[11] S.Fluhrer, I.Mantin, A.Shamir, Weakness in the key scheduling algorithm of RC4, 8th Annual Workshop on Selected Areas in Cryptography, August 2001.
[12] National Institute of Standards and Technology, "Specification for the Advanced Encryption Standard (AES)", Federal Information Processing Standards (FIPS) 197, November 2001.
Institute of Electrical and Electronics Engineers, "Local and Metropolitan Area Networks: Port-Based Network Access Control", IEEE Standard 802.1X, September 2001.
[13] Struif, B.; Scheuermann, D, "Smartcards with biometric user verification", Multimedia and Expo, 2002. ICME '02. Proceedings. 2002 IEEE International Conference on , Volume: 2 , 26-29 Aug. 2002 Pages:589 - 592 vol.2
[14] Gilbert, H., "The Security of One-Block-to-Many Modes of Operation", FSE 03, Springer-Verlag LNCS 2287, 2003.
[15] Iwata, T. and K. Kurosawa, "OMAC: One-Key CBC MAC", FSE 03, Springer-Verlag LNCS 2887, 2003.
[16] M.Loutrel, P.Urien, G.Pujolle, "A smartcard for authentication in WLANs", Proceedings of the 2003 IFIP/ACM Latin America conference on Towards a Latin American agenda for network research, La Paz, Bolivia, October 2003
[17] P.Urien, M.Loutrel,"The EAP smartcard. A tamper resistant device dedicated to 802.11 wireless networks", 3rd Worshop on applications and Services in Wireless Networks, Berne, Switzerland, July2-4, 2003.
[18] Institute of Electrical and Electronics Engineers, "Approved Draft Supplement to Standard for Telecommunications and Information Exchange Between Systems-LAN/MAN Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Specification for Enhanced Security", IEEE 802.11i-2004, 2004.
[19] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J. and H.Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
[20] Bellare, M., Rogaway, P. and D. Wagner, "The EAX mode of operation", FSE 04, Springer-Verlag LNCS 3017, 2004
[21] Urien P, Farrugia F, Groot M, Abellan J, "EAP-Support in Smartcard", draft-urien-eap-smartcard-08.txt , 2005
[22] Bersani.F, "The EAP-PSK Protocol: a Pre-Shared Key EAP Method", IETF draft, draft-bersani-eap-psk-06, 2004
[23] Renaudin, M.; Bouesse, F.; Proust, Ph.; Tual, J.P.; Sourgen, L.; Germain, F.; "High security smartcards", Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings , Volume: 1 , 16-20 Feb. 2004
[24] R.Brandewie, "Smart cards:world passport to security -identity solutions for a complex world." e-Smart 2004, Sept 22-24, 2004, Sophia Antipolis, Nice, France
[25] "Belgium electronic identity card (eID)". http://eid.belgium.be
[26] Timothy M. Jurgensen, Scott B. Guthery, "Smart Cards: The Developer's Toolkit", PRENTICE HALL
[27] OpenEapSmartcard WEB site, http://www.infres.enst.fr/~urien/openeapsmartcard