OpenEAP Smartcard.NET

1. Summary.

The goal of this project is to release a *dotnet* smartcard providing authentication services for network resources such as PPP, Wi-Fi and VPN (Virtual Private Network).
The Extensible Authentication Protocol is an IETF standard[1] which is widely used in personal computers. It is a flexible framework, that supports multiple authentication scenari such as
- EAP-TLS[2], a quite transparent transport of the well known SSL protocol.
- EAP-SIM[3] (or EAP-AKA[4]), an extension of SIM (or USIM) services for Wi-Fi infrastructures. As an illustration this authentication method is used in the emerging UMA[5] architecture, a VoIP service over Wi-Fi, in order to establish a secure (IPSEC) tunnel with an operator gateway.

An open software, OpenEapSmartcard[6] was previously released for javacards. OpenEapSmartcard.NET is an adaptation of this earlier work to dotnet smartcards.

Due to Cryptoflex.NET facilities, the OpenEapSmartcard.NET card offers two kind of interfaces
- A classical ISO7816 interface, which enables the deployment of this trusted device in existing software environments (an EAP DLL[7] working with smartcards) that cooperate with smartcards through legacy APDUs.
- An API interface that allows to any .NET developer, to transparently import the OpenEapSmartcard.NET highly secure services.

In this first prototype, two authentication methods are supported. The first is a simple, but still working, one way authentication method (based on the SHA1 algorithm), developed for education purposes. The second is more complex and is an open implementation of the EAP-TLS method, that fully processes the TLS protocol and that autonomously manages an embedded certificates store.

The OpenEapSmartcard.NET card is working in XP platform, it may be used in every network connection dealing with EAP, for advanced security features.

A demonstration is available in which the OpenEapSmartcard.NET device is used as an authentication token, controlling the access to a Wi-Fi network.

2. OpenEapSmartcard.NET services

The EAP smartcard services are described by an internet draft[8], whose twelfth version was issued in March 2006., they are classified in four categories,

---

[1] RFC 3748, 2004. Extensible Authentication Protocol, (EAP). Internet Engineering Task Force, IETF.
[2] RFC 2716, 1999. PPP EAP TLS Authentication Protocol. Internet Engineering Task Force, IETF.
[3] RFC 4186, " Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM) ", 2006
[4] RFC 4187, 2006. Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA); Internet Engineering Task Force, IETF
[5] Unlicensed Mobile Access (UMA), http://www.umatechnology.org
[6] http://www.enst.fr/~urien/openeapsmartcard
[7] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/eap/eap/about_extensible_authentication_protocol.asp
[8] http://www.ietf.org/internet-drafts/draft-urien-eap-smartcard-11.txt

- **The identity service**. A smartcard manages several network accounts; the terminal operating system performs an identity discovery process in order to browse its contents.
- **The Network service**. EAP messages are processed by the smartcard. At the end of the authentication method, a session key is computed.
- **The security service**. This service essentially manages PIN codes (Personal Identification Number) that are needed for security purposes.
- **The personalization service**. This service updates information stored in the smartcard.

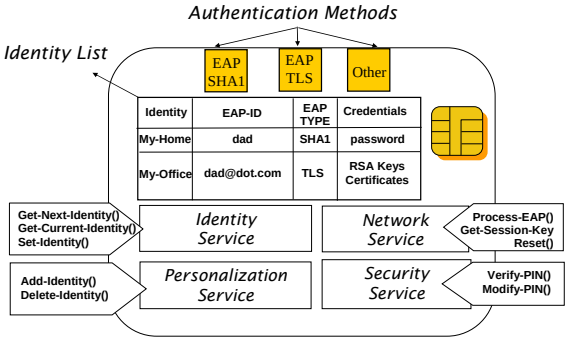

Figure 1. OpenEapSmartcard.NET services definition

2.1 Identity Service.

An EAP authentication scenario works with a set of three parameters,

- the EAP-ID, a user's identifier, transmitted in the *EAP-Identity.Response* message.
- the type (EAP-Type) of the authentication protocol (EAP-MD5, EAP-SIM, EAP-TLS....)
- a set of cryptographic credentials (shared secret, X509 certificates, RSA keys…) used by the authentication method.

Identity is a pointer to an authentication triplet, similar to the primary key of a relational table. The *Get-Next-Identity* command extracts an identity from a circular list. The operating system discovers all available identities, and selects one of them, or prompts a choice to the user. The *Set-Identity* command fixes the current identity managed by the card. Other primitives, like *Get-Current-Identity* provide additional facilities.

2.2 Network Service.

An EAP message (request or notification) is encapsulated in the *Process-EAP* command. The software which manages a state machine according to the selected type, delivers if necessary an EAP response message. At the end of the authentication protocol a session key (Master Session Key, MSK) is computed and read through the *Get-Session-Key* request.

2.3 Security Service

The embedded application manages two PIN codes, one is hold by the card bearer and the other by the card issuer. For example if the user's PIN is activated, the smartcard is locked (and can't be used) after three wrong PIN values presentation. PIN management facilities are similar to those described in the GSM 11.11 specification.

2.4 Personalization Service.

The *Add-Identity* and *Delete-Identity* commands personalize EAP applications or modify embedded credentials.

3. OpenEapSmartcard.NET integration in .NET platforms

The win32 platform introduced the notion of *EAP Provider*, e.g. a software dynamic library (DLL) that implements EAP authentication methods. We developed an *EAPCARD.DLL* component that is freely available on the *OpenEapSmartcard* WEB site. The Remote Access Service entity (RAS) manages all resident *EAP provider* objects.

-Upon system boot (1), the EAP provider object is invoked via the method *RasEapGetInfo* that returns three pointers to additional procedures named, *RasEapBegin*, *RasEapMakeMessage* and *RasEapEnd*.
- Human user interacts with the EAP provider through a dialog box, started by the *RasEapInvokeConfigUI* procedure. In our experimental implementation this graphical interface is used to get the smartcard identity list and to select the appropriate one.
- When the operating system detects a wireless cell, identified by its SSID, it performs the association process with the access point, sends an EAP-Start frame and activates the EAP provider that had been previously associated to this particular SSID. Then it calls the *RasEapGetIdentity* (2) method, which in turn, sends an identity request message to the EAP smartcard.
- When the EAP identity request message is received, an EAP session is started and the system calls the *RasEapBegin* (3) function. At this point the smartcard application is selected and the bearer enters his PIN code.
-During an EAP session all EAP requests or notifications are sent to the *RasEapMakeMessage* (4) method, which forwards them to the EAP smartcard.
- At the end of the authentication scenario the *RasEapEnd* (5) method is invoked.

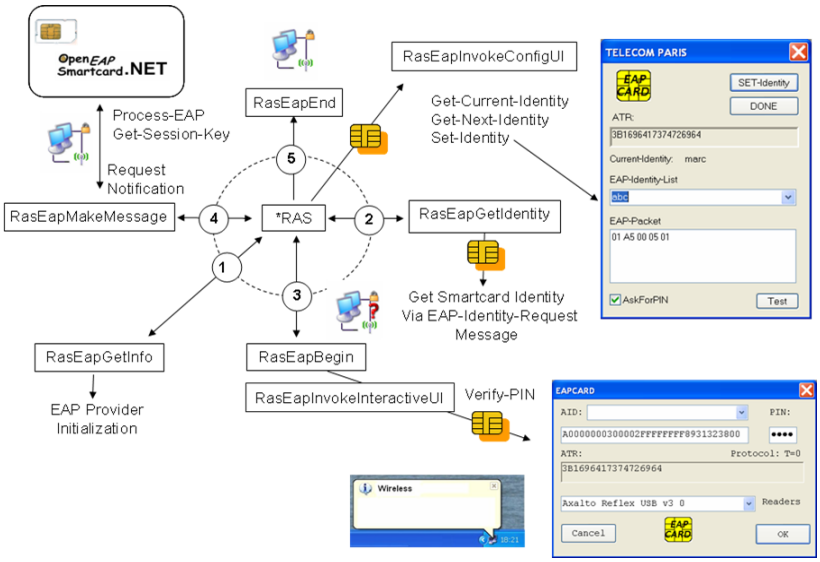

Figure 2. OpenEapSmartcard.NET interaction with a dedicated EAP provider DLL

4. OpenEapSmartCard.NET Architecture

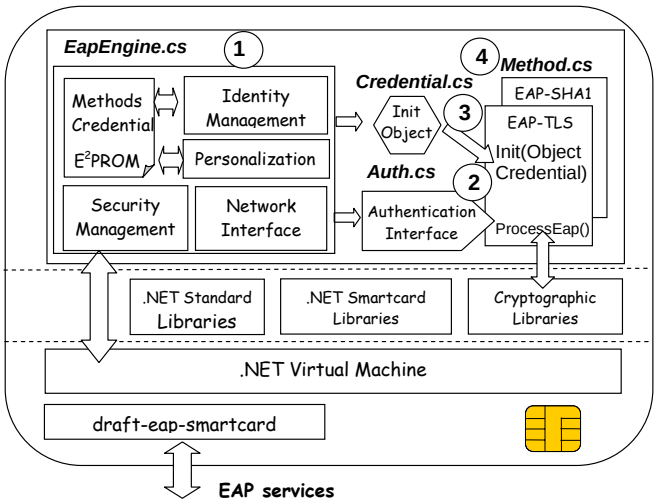

Figure 3. OpenEapSmartcard.NET Architecture

The software architecture mainly comprises four .NET components,

1- The *EapEngine* which implements the EAP core, and acts as a router that sends and receives packets to/from authentication methods
2- An *Authentication Interface* that defines all services offered by EAP methods
3- A *Credential Object* which stores information needed for method initialization.
4- One or more *Methods* that instantiate authentication scenari like EAP-TLS or EAP-SHA1

4.1 EapEngine

This object 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 which is collected by the terminal operating system.

4.2 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 (*fct()*) dedicated to performances evaluations.

4.3 Credential Objects

Every method is associated to a *Credential Object* that encapsulates all information required to process a particular authentication scenario (shared secret, X509 certificates, RSA keys,…)

4.4 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.

5. Implementation details

5.1 Classes diagram.

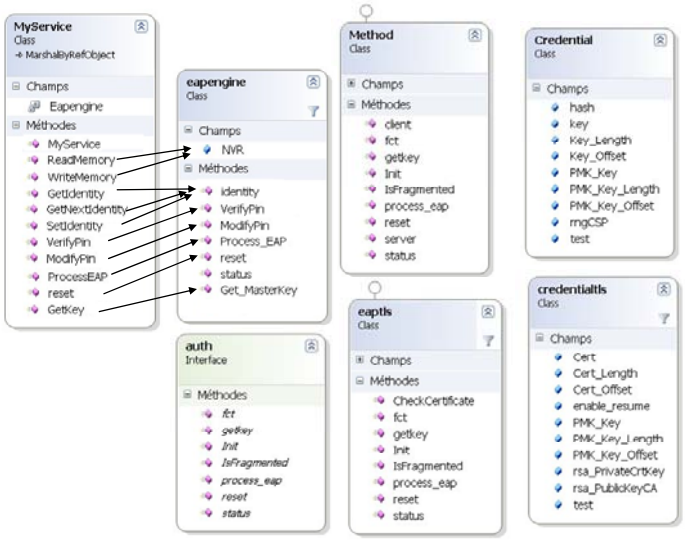

Figure 4. Classes diagram of the OpenEapSmartcard.NET software implementation

The embedded software comprises the eapengine (*eapengine.cs*), the authentication interface (*auth.cs*), and two authentication methods

- *Method.cs* and its associated credential object (*Credential.cs*) realizes a simple one way authentication scenario based on the SHA1 algorithm. This class comprises the EAP client entity (client) and the EAP server entity (server). This allows to perform a complete dialog between server and client parts, within a single smartcard.
- *eaptls.cs* and its associated credential object (*credentialtls.cs*) is a standalone implementation of the EAP-TLS method. Certificates are verified thanks to the *CheckCertificate* procedure. With this first implementation, which is not yet optimized, the processing of a full EAP-TLS authentication scenario costs about ***18 seconds***.

5.2 Remote services.

5.2.1 API Interface

The "openeapdotnet.uri" service supports the following procedures:

- *public bool VerifyPin(byte[] pin),* checks the user's PIN code
- *public bool ModifyPin(byte[] oldpin, byte[] newpin),* modifies the user's PIN code
- *public byte[] ReadMemory(short offset, short length),* reading of the smartcard memory that stores the bearer's credentials.
- *public bool WriteMemory(short offset,byte[] data),* writing of the smartcard memory that stores the bearer's credentials.
- *public byte[] GetNextIdentity(),* gets the next available identity managed by the smartcard
- *public byte[] GetIdentity(byte[] idt),* retrieves the current identity.
- *public byte[] SetIdentity(byte[] idt),* sets the new identity
- *public bool reset(),* resets the current authentication method.
- *public byte[] GetKey(),* collects the master session key, upon a successful authentication.

- *public byte[] ProcessEAP(bool more,byte[] msg)*, processes an EAP packet, returns an EAP message.

5.3 Legacy ISO7816 interface

The "A0000000300002FFFFFFFF8931323800" service supports the following procedures,

[APDU("A0200000", Mask = "F000FFFF")],
public void VerifyPin(…), checks the user's PIN code.

[APDU("A0240000", Mask = "F000FFFF")]
public void ChangePin(…), modifies the user's PIN code.

[APDU("A0B00000", Mask = "FC00FFFF")]
public byte[] Read(…), reading of the smartcard memory that stores the bearer's credentials.

[APDU("A0D00000", Mask = "FC00FFFF")]
public void Write(…), writing of the smartcard memory that stores the bearer's credentials.

[APDU("A0170001", Mask = "F000FF00")]
public byte[] GetNextIdentity(…), gets the next available identity managed by the smartcard

[APDU("A0180000", Mask = "F000FF00")]
public byte[] GetIdentity(…), retrieves the current identity.

[APDU("A0160080", Mask = "F000FF00")]
public byte[] SetIdentity(…), sets the new identity

[APDU("A0191000", Mask = "F00000FF")]
public void Reset(…), resets the current authentication method.

[APDU("A0A60000", Mask = "F000FFFF")]
public byte[] GetKey(), collects the master session key, upon a successful authentication.

[APDU("A0800000", Mask = "F000FFFF")]
public byte[] ProcessEAP(…), processes an EAP packet, returns an EAP message.

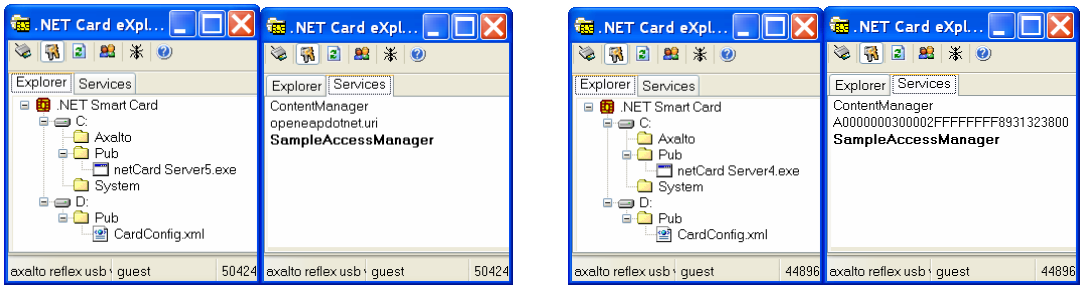

Figure 5. OpenEapSmartcard services using, APIs (*left side*) or legacy APDUs (*right side*)

6. Demonstrations

6.1 API interface

The smartcard service is named *server5.exe*, the .NET program is called *console3.exe* and tests the remote used of EAP services (EAP-SHA1 and EAP-TLS) offered by the OpenEapSmartCard.NET device.

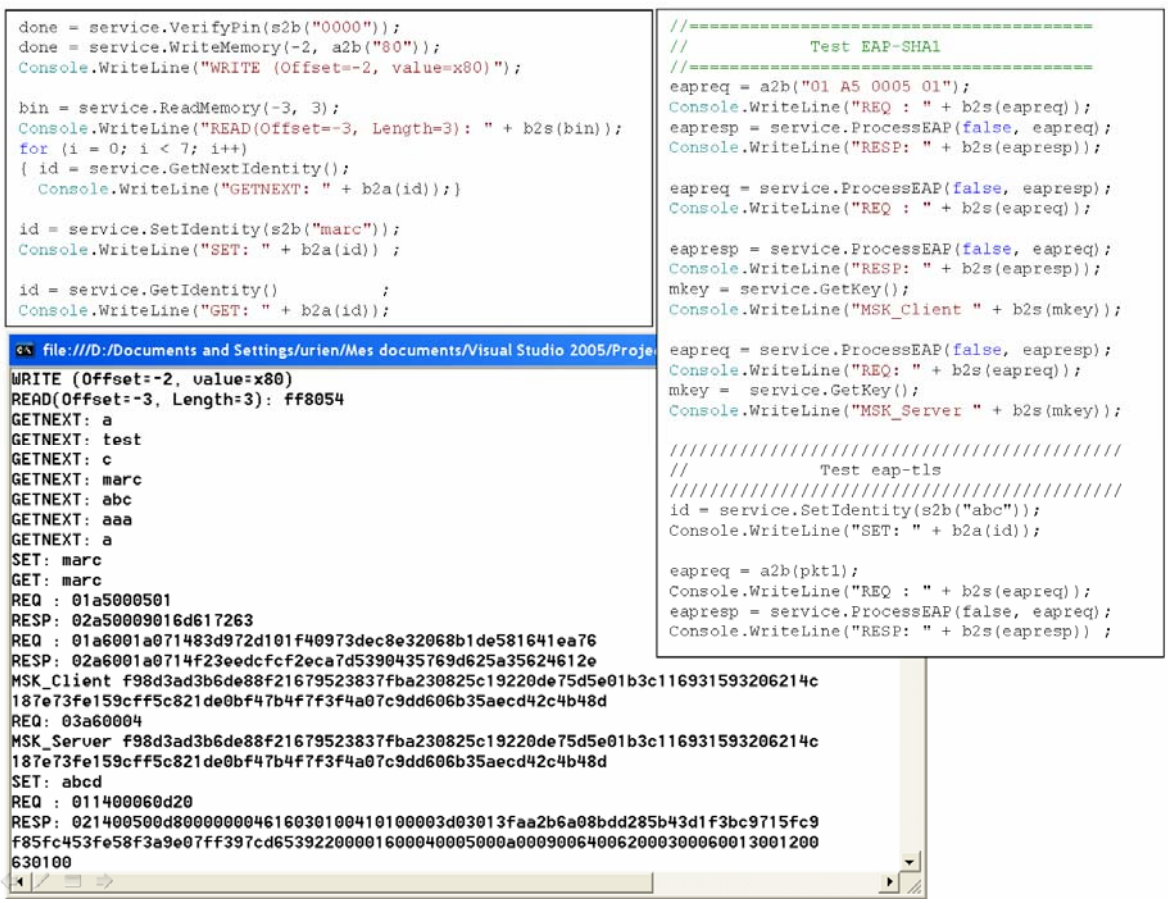

Figure 6. Basic tests of the OpenEapSmartcard.NET services

6.2 Wi-Fi deployment

The smartcard service is named server4.exe, and is associated to the AID "A0000000300002FFFFFFFF8931323800". In this demonstration the smartcard works with an associated DLL (eapcard.dll) that is not written in .NET. Therefore this component communicates with the smartcard via legacy ISO7816 commands.

The OpenEapSmartcard.NET device processes all EAP message and computes the *MPPE-Recv-Key* and *MPPE-Send-Key* that are collected by the operating system in order to ensure the radio security.

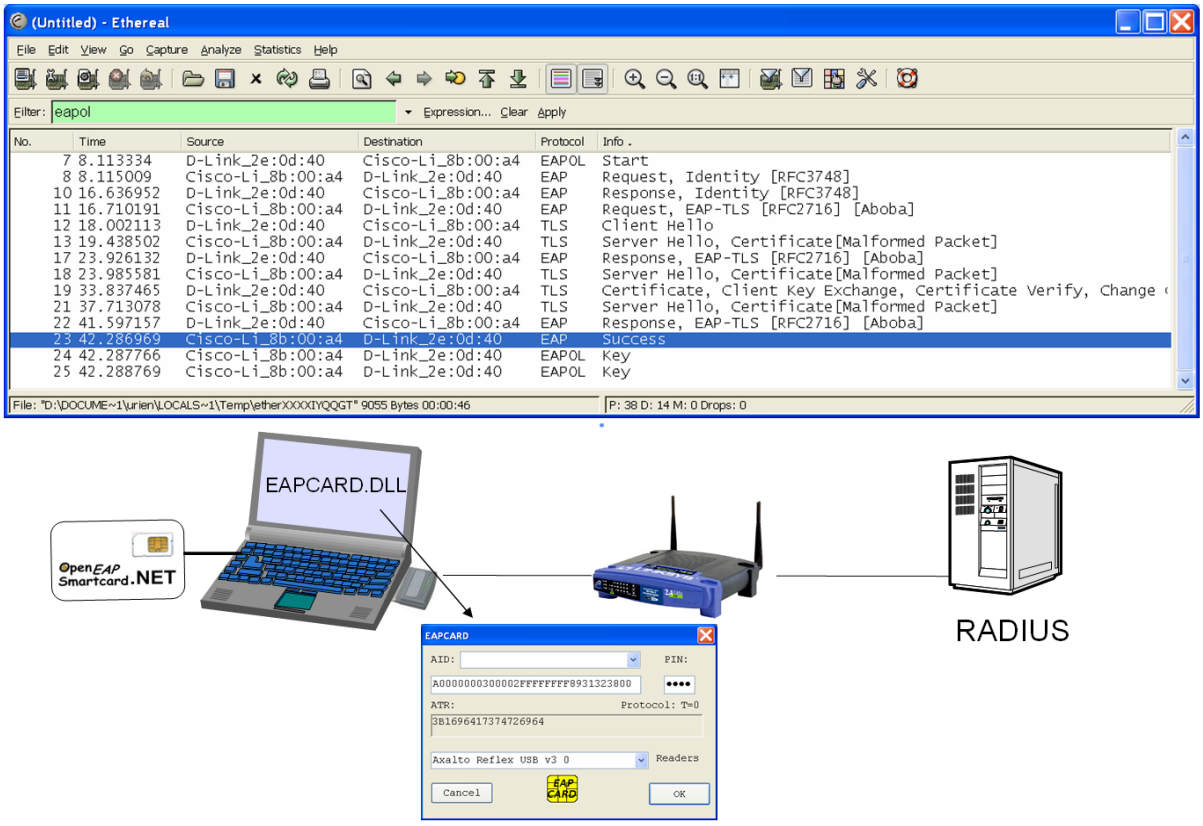

Figure 7. OpenEapSmartcard.NET demonstration in a Wi-Fi environment.

7. Conclusion and further work

This work is the first release of the *OpenEapSmartcard.NET* software that autonomously processes EAP methods in *dotnet* smartcards. Although it is not yet optimized, performances are sufficient for real Wi-Fi networks. The EAP-TLS support, in a tamper resistant device, introduces the notion of the *pocket PKI*, that manages an embedded certificate store, and that safely computes the TLS protocols and its associated private keys.