Framework Implementation Based on Grid of Smartcards to Authenticate Users and Virtual Machines

Hassane Aissaoui-Mehrez[1], Pascal Urien[1] and Guy Pujolle[2]
[1]*Institute of Mines-Telecom / TELECOM-ParisTech: LTCI CNRS Laboratory*
Network and Computer Science Department, 46 rue Barrault, 75634 Paris, France
[2]*University of Pierre and Marie Curie, CNRS LIP6/UPMC Laboratory 4 Place Jussieu, 75005 Paris, France*

Keywords: OpenID, Microcontrollers, Secure Elements, User-Centric Identity, Virtualization and Cloud Computing.

Abstract: The Security for the Future Networks (SecFuNet) project proposes to integrate the secure microcontrollers in order to introduce, among its many services, authentication and authorization functions for Cloud and virtual environments. One of the main goals of SecFuNet is to develop a secure infrastructure for virtualized environments and Clouds in order to provide strong isolation among virtual infrastructures, and guarantee that one virtual machine (VM) should not interfere with others. The goal of this paper is to describe the implementation and the experimentation of the solution for identifying users and nodes in the SecFuNet architecture. In this implementation, we also employ low-cost smartcards. Only authorized users are allowed to create or instantiate virtual environments. Thus, users and hypervisors are equipped with secure elements, used to open TLS secure channels with strong mutual authentication.

1 INTRODUCTION

The IdM system will be based on smartcards OpenID and user-centric attribute control policies. The authentication servers are composed by secure microcontrollers.
The objective is to implement the framework, based on the authentication servers and EAP-TLS smartcard model. The proposed SecFuNet framework provides TLS secure channels for establishing trust relationships among Users, VMs, XEN and Grid of Secure Elements (GoSE). The authentication is done directly between smartcards (owned by users or associated to VM) and a GoSE arranged in a SecFuNet IdP.

This paper concerns a highly secure authentication server with an array of secure microcontrollers allowing users' or VMs' strong mutual authentication with GoSE. It defines the structure and the components of the authentication server.
This paper is organized as follows. Section 2 presents a related work and a brief state-of-art of EAP-TLS Smartcard Concept and Software Architecture. The next section 3 details the use of TLS-Id based on EAP-TLS smartcard in OpenID platforms. In the section 4, we detail how dedicated Service Provider may be used to download tokens in EAP-TLS smartcard?

2 RELATED WORK

To address some of the security issues, the project aims to explore the application of secure elements, such as smartcards, to improve the trustworthiness of network infrastructure services for future networks. Two classes of secure microcontrollers have been studied, smartcards and TPMs (Trusted Platform Modules).

These electronics chips have different computing capabilities, smartcards usually run a Java Virtual Machine (JVM) and therefore are able to execute complex procedures (such as the TLS protocol), while TPMs are dedicated to the RSA algorithm. However these devices may be used in order to enforce trust for the TLS protocol or to guarantee secure storage for cryptographic keys.

These security properties are directly provided by smartcards (thanks to dedicated embedded software), but require additional software components for TPMs.

Aissaoui-Mehrez H., Urien P. and Pujolle G..
Framework Implementation Based on Grid of Smartcards to Authenticate Users and Virtual Machines.
DOI: 10.5220/0005046102850290
In *Proceedings of the 11th International Conference on Security and Cryptography* (SECRYPT-2014), pages 285-290
ISBN: 978-989-758-045-1
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

The main goal of this section is to briefly overview the context of the virtualization platforms and EAP-TLS smartcard model.

2.1 Extensible Authentication Protocol (EAP) and TLS

The EAP (RFC 3748, 2004) is a flexible framework targeting access control in various network infrastructures such as classical LAN, wireless LAN, or VPNs. Furthermore the EAP-TLS (RFC 5216, 2008) standard enables a transparent transport of TLS in EAP messages; it offers a convenient way to use TLS without TCP flavours, and provides mechanisms for TLS packets segmentation and reassembly. The SecFuNet TLS platform is based on EAP-TLS smartcards, equipped with additional facilities. These devices include a full TLS stack, an entity managing X509 certificates, and provide a secure environment for keys storage and cryptographic procedures execution. The TLS master stack forwards packets, which are thereafter encapsulated in EAP-TLS messages "Figure 1". When it detects the completion of this phase (trigged by finished messages) it uses two ISO7816 commands (APDUs) in order to retrieve CipherSuite and KeyBlock parameters.

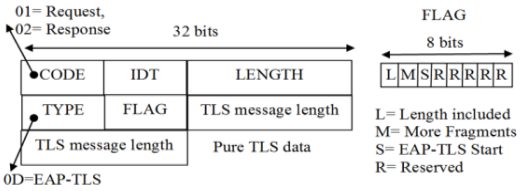

Figure 1: The EAP-TLS Header.

2.2 EAP-TLS Smartcard Concept

In this section we focus on EAP-TLS smartcard and TLS Identity (TLS-Id) deployed in SecFuNet. A smartcard (Jurgensen, 2002) is a tamper resistant device, including CPU, RAM and non volatile memory.

Most of the electronic chips (i.e. smartcard) support a JVM and execute software written in this programming language (Chen, 2002).

The use of smartcards in TLS authentication has now a rather long history and has been largely developed according to different models.

These devices run the JAVA open stack, introduced in (Menon, 2006) and which comprises four logical components. The software architecture of the EAP-TLS smartcard (Pujolle, 2008)(Urien, 2013) is the following:

- An EAP engine, which implements four fundamental services (EAP messages treatment, identity management, security functions, and personalization) and ensures EAP routing towards authentication methods supported by the card.
- EAP-TLS method, which manages fragmentation and reassembly mechanisms.
- TLS stack. The Handshake protocol takes responsibility of authentication mechanisms, whereas the record protocol realizes the encryption and the integrity of data securely transported by the TLS tunnel.
- A certificates store.

The TLS-Identity (TLS-Id) is a set of five parameters, which comprises:

- X509 certificate and its associated private key;
- Certification Authority certificate;
- EAP identity parameter (EAP-ID);
- Friendly name, used to identify and to activate a given TLS-Id.

These parameters are embedded in the secure microcontroller hold by SecFuNet users, during the personalization process.

2.3 TLS Choreography with EAP-TLS Smartcard

The TLS-Tandem card is plugged to a SecFuNet host, such as personal computer or mobile handset. Before opening a session with a TLS server, its electronic identity is activated via the Set-Identity command. The host intends to download a file, typically through an HTTP request, which is securely stored in a remote WEB server "Figure 2".

The docking host manages TCP/IP operations and opens a connection with a remote TLS server, and then calls the connection procedure.

This latter resets the TLS-Tandem card, and sends an EAP-TLS-Start packet concatenated to Unix-Time; the smartcard produces a response including the first TLS message. A software component acts as a bridge between EAP-TLS and TLS.

It removes EAP-TLS header (typically 10 bytes), and sends TLS messages to remote server. It reads incoming data, detects TLS error, and determines the end of TLS messages such as:

- ServerHelloDone that indicates a four ways handshake associated to a full session.
- Finished that indicates a three ways handshake associated to a resume session.

Thereafter it packs this set of TLS messages in a single EAP-TLS packet, appends a 6 bytes prefix, and forwards this request to the card.

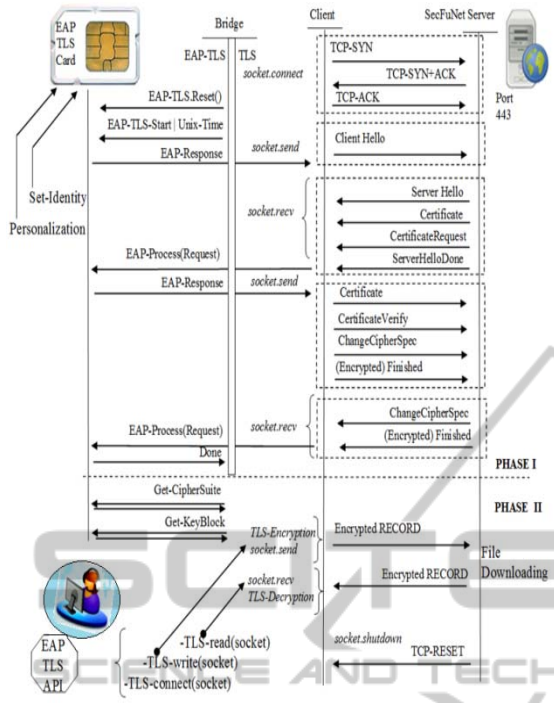

Figure 2: A TLS session dealing with EAP-TLS smartcard.

The card generates a response which is processed by the EAP-TLS/TLS bridge and transmits to the server.

Upon, the docking host collects two parameters from the smartcard: CipherSuite and Key-Block. The procedure is over. Thereafter smartcard is not used, and the docking host encrypts and decrypts information, which are sent and received to/from remote server. These operations are done via the TLS-write and TLS-read() functions.

3 EAP-TLS SMARTCARD AND OpenID

OpenID is a Single Sign On system made of three entities, the consumer site requiring a user authentication, the OpenID server performing the user authentication, and an internet user equipped with a terminal. The OpenID identifier is an URL that comprises two parts the name of the OpenID server and the user alias.

There are many ways to perform authentication between the user and the OpenID server. The most popular is the simple password mechanism.

In the SecuFuNet context we use a strong mutual authentication based on a TLS session running in the EAP-TLS device, in which all resources (i.e. client, VM, server…) are identified by their X509 certificates and associated private keys.

3.1 Service Discovery and Shared Secret Computing

In the OpenID context the user identity is associated to an URL. The SecFuNet user equipped with a secure element, logs to the Service Provider (SP) WEB site, where he gives his identity and indicates the solicited service. The SP deduces from the user's identity the OpenID server name. The consumer starts a discovery (XRI) procedure with the provider and collects the XRDS server address. These two entities exchange thereafter resource descriptors encoded according to the XRDS format.

Afterward the server and the consumer perform a classical Diffie-Hellman key exchange (DH) of their public keys, in order to compute a symmetric shared secret used to enforce message integrity thanks to a HMAC procedure.

3.2 Authentication with the SEs

The OpenID authentication server receives the Authentication Request, as an HTML POST message issued by the user's terminal. It thereafter returns an HTML form "Figure 3".

In a classical provider implementation, the user would be identified by a password collected by this form, the login page being protected via a TLS session. In the SecFuNet platform we suppress this password, and we replace it by a TLS session with mutual authentication (i.e. both sides hold X509 certificate and RSA private key), initiated by the EAP-TLS secure element.

The login page is associated to an HTTP header including a cookie whose value is the session identifier (sid), and the associated URL also comprises this value, i.e. looks like:

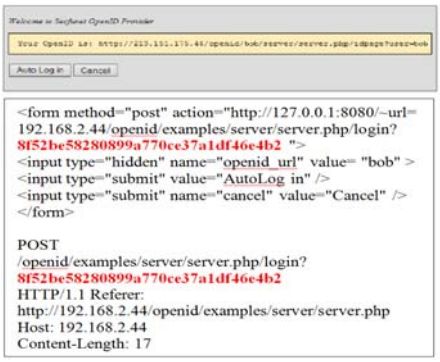

Figure 3: OpenID form used for user authentication.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

http://127.0.0.1:8080/~url=server.com/login.php?sid

When the user clicks on the AutoLogin button it opens via its EAP-TLS token the https//server.com/login.php?sid link, and is consequently authenticated by its X509 certificate.

The HTTP response to this request will set the cookie (sid) for the proxy address (dealing with the loopback address such as 127.0.0.1:8080).

A second optional exchange occurs between the provider and the browser (secured by the EAP-TLS token), in order to confirm the association with the consumer site. This step may be used for downloading attributes in the EAP-TLS secure element.

3.3 Authentication Response

After a successful authentication, the Authentication Response is returned by the OpenID server to the user's terminal. It is thereafter redirected to the consumer site, which finally delivers a welcome page. As detailed "Figure 4" an Authentication Response message is delivered by the SP.

It is a set of response parameters included in location MIME header. Data integrity is enforced by an HMAC field associated with the OpenID secret key, computed from the previous DH exchange.

```
HTTP/1.1 302 Found
Date: Sun, 17 Mar 2014 09:07:28 GMT
Server: Apache/2.2.6 (Win32) DAV/2 mod_ssl/2.2.6 OpenSSL/0.9.8g mod_autoindex_color
PHP/5.2.5
X-Powered-By: PHP/5.2.5
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Thu, 19 Nov 2015 08:52:00 GMT
location: http://192.168.2.35:80/openid/examples/consumer/finish_auth.php?
janrain_nonce=2014-03-17T09:07:14ZGoN7wY
&openid.assoc_handle={HMAC-SHA1}{49bf5e64}{Va0OCQ==}
&openid.claimed_id=http://192.168.2.44/openid/examples/server/server.php/idpage/user=bob
&openid.identity=http://192.168.2.44/openid/examples/server/server.php/idpage/user=bob
&openid.mode=id_res
&openid.ns=http://specs.openid.net/auth/2.0
&openid.ns.sreg=http://openid.net/extensions/sreg/1.1
&openid.op_endpoint=http://192.168.2.44/openid/examples/server/server.php
&openid.response_nonce=2014-03-17T09:07:28Z2Z6LCL
&openid.return_to=http://192.168.2.35:80/openid/examples/consumer/finish_auth.php/
janrain_nonce=2014-03-17T09:07:14ZGoN7wY
&openid.sig=HUBr4K7Ff51FHCywZFux2IcZ9Xg=
&openid.signed=assoc_handle,claimed_id,identity,mode,ns,ns.sreg,op_endpoint,response_nonce,
return_to,signed,sreg.email,sreg.fullname,sreg.nickname
&openid.sreg.email=bob@secfunet.com
&openid.sreg.fullname=Bob+FirstName
&openid.sreg.nickname=bob
Connection: close
Content-Length: 0
Content-Type: text/html
```

Figure 4: OpenID Authentication Response Message.

4 LOADING CRYPTOGRAPHIC TOKENS IN SMARTCARD

The goal of this section is to detail how dedicated SP may be used to download tokens in EAP-TLS smartcard. The user attributes instantiated by tokens are stored in a secure element. Two use cases are described in order to load cryptographic token in smartcard: the use of a classical WEB interface to authenticate user with its EAP-TLS Secure Element based on the AJAX technology, and the use of GoSE operations to authenticate VM.

4.1 User Authentication Using SE

As described in previous sections the SecFuNet user is equipped with an EAP-TLS secure element "Figure 5". In order to collect tokens, it performs a connection with the appropriate service provider. He enters his identity and is thereafter redirected to "adhoc" OpenID server. The server forwards a login page and an authentication is performed by the EAP-TLS secure element. Afterwards the OpenID server delivers a page that interacts with the EAP-TLS secure element. The interaction between the server and the EAP-TLS Secure Element is based on the Asynchronous JavaScript and XML (AJAX).

AJAX is a set of technologies for executing applications on the browser side, triggered by user's interactions, typically mouse clicks associated with HTML forms, and processed by JavaScript procedures. AJAX pages are downloaded by the user's terminal. The user's terminal is running a TCP daemon that we call the Proxy Server (PS), opened on the loopback address (127.0.0.1), usually with the port 8080. Its interface is made by two kinds of URLs, used for network exchange and EAP-TLS secure element.

The first class of URLs (Class I), such as: http://127.0.0.1:8080/~url=www.server/path/file.html, opens an HTTPS session the remote server (i.e. is equivalent to https://www.server/path/file.html).

The TLS session is booted from the EAP-TLS secure element and then transferred back to the PS. As a result the browser opens HTTP session with remote servers that are authenticated via the USIM. The second class of URLs (Class II), such as:

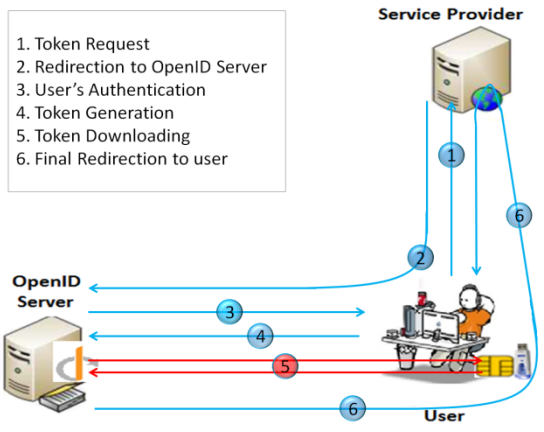

Figure 5: User token generation and downloading.

http://127.0.0.1:8080/reader/apdu.xml, sends ISO7816 commands to the EAP-TLS secure element. These commands are located in the body of an HTTP POST request. There are encoded as the content of form inputs: =CMD1&=CMD2&=CMDi. The returned response is encoded according to the XML format. AJAX pages returned by the OpenID server include XMLHttpRequest objects that send Class II HTTP requests to the proxy server.

These requests transport ISO7816 commands, executed afterwards by the EAP-TLS secure element. The proxy returns ISO7816 responses, embedded in XML pages. These XML contents are parsed and thanks to the DOM (Document Object Model for JavaScript) the initial page is modified, typically with a success status.

"Figure 6" illustrates AJAX interaction with the EAP-TLS secure element. An ISO7816 command is hidden in an HTML form; it is thereafter executed by the proxy server.

Thanks to these mechanisms the OpenID server pushes content in the EAP-TLS secure element. During the authentication procedure the OpenID server collects the certificate of the EAP-TLS secure element, which contains a public key.

This cryptographic material may be used to build a container, able to securely convey token to be downloaded in the EAP-TLS secure element.

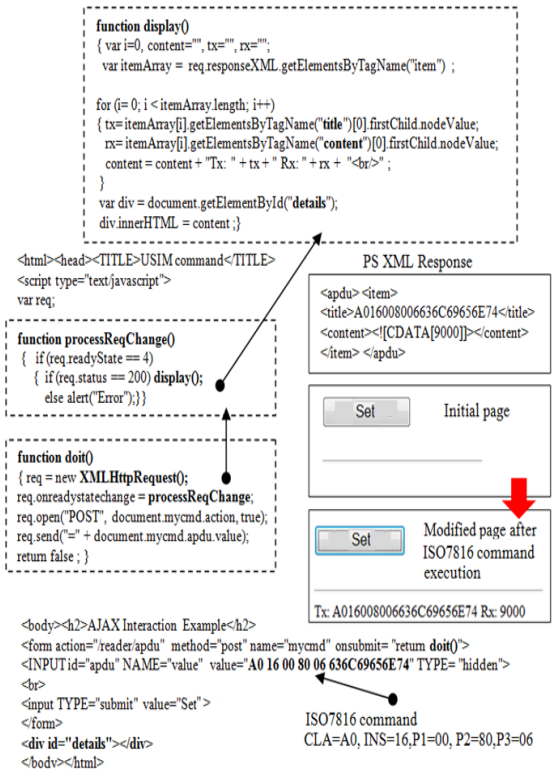

Figure 6: Interaction between EAP-TLS SE and OpenID.

A set of ISO7816 commands, embedded in AJAX pages, realizes operations needed for container downloading. A container is made of three parts:

- A header, which is the encrypted value of a symmetric AES key, with the EAP-TLS secure element public key, according to the PKCS#1 standard.
- A body, which is the encrypted value of the token, according to an AES-CBC procedure.
- A trailer, which is the signature by a trusted authority (identified by its public key) of the header concatenated to the body, according to the PKCS#1 standard.

Upon downloading, the signature of the container is checked; the AES key is recovered from the EAP-TLS private key, the token is afterwards decrypted and stored in the smartcard.

4.2 VM Authentication Using GoSE

As previously mentioned, each VM is associated to an identifier (VM_{ID}, i.e. a certificate), and the VM Authorization Token (VM-Auth-Token) establishing the link with the hypervisor (identified by its identifier HV_{ID}) that hosts the VM. Each VM is also associated to a secure element, plugged in a grid, which securely stores its private key.

In order to remotely work with its private key, the VM must interact the service provider that interface the GoSE. The VM is authenticated by the VM_{ID}, and the hypervisor private key and VM-Auth-token. This use case is illustrated by "Figure 7".

The VM may access to the private key of the hypervisor stored in a secure element. It establishes a session with the grid service provider, and provides the HV_{ID}, VM_{ID} and VM-Auth-Token parameters. It also request cryptographic operation with the GoSE, such as encryption or decryption with the VM private key. It is thereafter redirected toward the OpenID server that performs the authentication process with the HV secure element.

Strictly speaking the OpenID authenticates the hypervisor, and it is not aware of an existing functional link with the VM.

At this step the service provider knows that the remote entity has access to the HV private key and is supplied with the appropriate credential (VM-Auth-Token). The GoSE service provider realizes the requested operation and returns the result.

Obviously the session between the VM and the GoSE service provider must be secured by the TLS protocol (i.e. HTTPS requests are used).

In order to get a greater security level, the GoSE service provider may pack the result in a container, which is made of three parts:

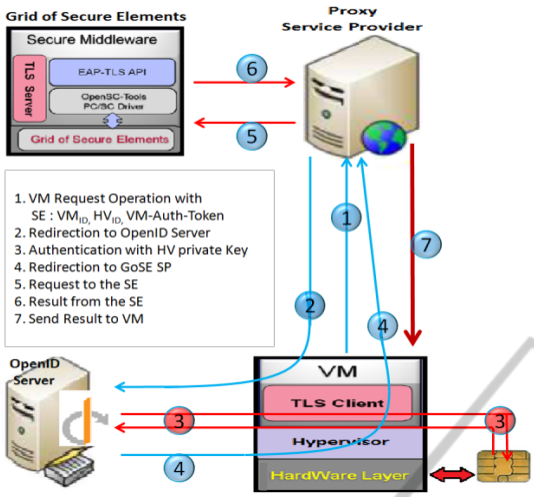

Figure 7: Grid SP operations in an OpenID context.

- A header, which is the encrypted value of a symmetric AES key, with the HV public key, according to the PKCS#1 standard.
- A body, which is the encrypted value of the result of the requested operation, according to an AES-CBC procedure.
- A trailer, which is the signature by a trusted authority (identified by its public key) of the header concatenated to the body, according to the PKCS#1 standard.

Some trust issue may exist between the grid SP and the VM, because they don't necessarily share the same ring of confidence. In that case the container could be directly generated by the VM secure element, previously bound to an HV_{ID}. This operation is performed by administrator of secure element thanks to Global Platform mechanisms.

TLS sessions with strong mutual authentication are mandatory for all virtual machines (VM). Thus, we implement the TLS Application in VM-side and the Secure Middleware in Xen-side, including a very simple facility to access GoSE, in order to enforce a remote VM access to secure element and to secure connections and transactions.

5 CONCLUSION

The architecture proposed and implemented in this deliverable presents a secure solution for user or VM connection in the context of the SecFuNet, based on grid of secure elements. This integration provides a higher level of security when compared to traditional password authentication, and establishes trust relationships among the resources, and eliminates the vulnerabilities like phishing attacks.

Strong authentication is done directly between smartcards (owned by user or associated to VM) and a GoSE.

We have seen in this paper that the use of secure elements can make a real highlight in the certificate management and the application of a security policy.

For large infrastructure, add our Secure Middleware brings the real simplicity of management. In parallel, can be very interesting solution to access securely to a remote specific area of GoSE.

The experimental results of the platform developed for SecFuNet demonstrates that the scalability performances are compatible with today constraints.

Furthermore, the smartcard shall be a great addition to virtual architecture and will be as well as a key asset to securing Cloud Computing infrastructures.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

ACKNOWLEDGEMENTS

This work has had financial support from CNPQ through process 590047/2011-6 (SecFuNet project) and also through processes 307588/2010-6 and 384858/2012-0. We also thank CAPES for the financial support with PhD scholarship.

REFERENCES

Jurgensen, T.M. et. al., 2002. Paper Prentice Hall PTR, ISBN 0130937304, Smartcards: The Developer's Toolkit.

Chen, Z., 2002. Addison-Wesley Pub Co 2002, ISBN 020170329, Java CardTM Technology for Smart cards: Architecture and Programmer's.

Menon, A., Cox, A. L., and Zwaenepoel, W., 2006. in Proceedings of the annual conference on USENIX '06 Annual Technical Conference, ATEC' 06, (Berkeley, CA, USA), pp. 2–2, USENIX Association, Optimizing network virtualization in xen.

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

Urien, P., 2013. IETF draft, EAP-Support in Smartcard", draft-urien-eap-smartcard-25.txt.

RFC 3748, 2004. Extensible Authentication Protocol, (EAP).

RFC 5216, 2008. The EAP-TLS Authentication Protocol.