Smart Card support Embedded Within OpenSSL to Secure Virtual Machines

Authors:

Hassane AISSAOUI-MEHREZ
Pascal URIEN
Guy PUJOLLE

Télécom Paris Tech Institute
Télécom Paris Tech Institute
Pierre & Marie Curie University
Introduction

• Security for Future Networks (SecFuNet) is a collaborative research project between Brazilian & European Commission.

• One of the main goals of this project is to develop a secure infrastructure for virtualized environments in order to provide strong isolation for virtual machines.

• How to secure VM communications?
• Using a TLS : is good idea!

• But where we store a sensitive keys of VMs?
• The solution is to use a Grid of Secure Elements (GoSE) for enforcing identity of Virtual Machines.

• The goal of this presentation is to describe the implementation of our solution for identifying users and nodes in the SecFuNet architecture based on OpenSSL.

• The authentication is done directly between smart card (owned by users, XEN Hypervisor, or associated to VM) and a SecFuNet Identity Provider.
• The GoSE is:

  – **SIM Array Server** hosting a set of secure elements (i.e.: smart cards).
  – All accesses to a GoSE require the TCP-IP transport.
  – Secured by the TLS protocol.
  – The SE is compatible with ISO 7816
What about a Grid of Secure Elements (GoSE)?

- The goal of this platform is to deliver Trusted as a Services (TaaS) over the Internet.

- Typically we can use it as a central SIM Server in distributed applications:
  - identifications of Virtual Machines,
  - isolation of Network on Demand (i.e. SDN).
What about a Secure Element (SE)?

- This electronic chip is smart card, supporting a Java Virtual machine (JVM).

- We embedded in each Smart card a new EAP-TLS Framework:
  - **EAP**: Extensible Authentication Protocol,
  - **TLS stack**: Transport Layer Security,

- The authentication method with TLS is performed in Secure Element.
• is as follow:

• **EAP-TLS** method: which manages fragmentation.

• **TLS stack**:
  – To provide Handshake protocol,
  – Record Layer protocol: to realize the encryption and the integrity of data.

• **Container**: to stores all keys (certificates, private keys, symmetric key...).
Where is OpenSSL in All This?
SDK-Smart-Card-API

Product Offerings

```c
/*
 * smart card interface for EAP-TLS Smart Card
 * (/openssl/scard_pesc_api/scardpesc.h
 */

typedef enum {
    SCARD_GSM_SIM_ONLY,
    SCARD_USIM_ONLY,
    SCARD_TRY BOTH,
    SCARD_EAPTLS_ONLY
} scard_sim_type;

typedef enum {
    SCARD_GSM_SIM, SCARD_USIM, SCARD_EAPTLS
} sim_types;

struct scard_data {
    SCARD_CONTEXT ctx;
    SCARDHANDLE card;
    DWORD protocol;
    sim_types sim_type;
    int pin1_required;
    unsigned long nbr_reader;
    unsigned char BufRecv[NAXTLSMSG];
    unsigned char BufSend[NAXTLSMSG];
    DWORD SendLen;
    DWORD RecvLen;
    struct wpabuf *CipherSuite;
    struct wpabuf *KeyBlock;
};

/*
 * Procedures for EAP-TLS Smart Card support
 */

int scard_make_eap_tls_msg_output(struct scard_data *scard, int fragments_size);

struct scard_data *scard_init_eap_tls(scard_sim_type sim_type, const char *reader, int pin_needed, unsigned long readertohandle);

int scard_select_file_eap_tls(struct scard_data *scard, sim_types sim_type, unsigned char *aid, size_t aidlen);

int scard_add_time_eap_tls_start(struct scard_data *scard);

int scard_set_identity(struct scard_data *scard, const char *identity);

int scard_reset_eap_tls(struct scard_data *scard);

int scard_get_ciphersuite(struct scard_data *scard);

int scard_get_keyblock(struct scard_data *scard);

struct scard_data * scard_init(scard_sim_type sim_type, const char *reader);

void scard_deinit(struct scard_data *scard);

int scard_verify_pin(struct scard_data *scard, const char *pin);

int scard_set_pin(struct scard_data *scard, const char *pin);

int scard_get_pin_retry_counter(struct scard_data *scard);
```
SDK-Smart-Card-API
Product Offerings

Proxy Client, Proxy VM & Proxy Server Procedures

Proxy RACs Procedures

Product Offerings
How it works?
How it works?

$ openssl s_scard
  -tls1
  -debug
  -connect remote_tls_server
  -port 443
  -pin_code

# ./openssl s_server
  -msg -debug -tls1
  -accept 443
  -verify 1
  -CAfile root.pem
  -cert tls_server_cert.pem
  -key tls_server_key.pem
  -pass pass:pwd
Use case I: Interactions between Proxy Client & SE

- The EAP-TLS smart card is plugged to a SecFuNet host such as: personal computer or mobile handset.

- The user’s terminal is running a Proxy as TCP daemon.

- The Proxy activated the electronic identity in smart card, via the Set-Identity command.

- The host intends to open a secure connection typically through an HTTPS request, with a remote SecFuNet WEB server.

- The docking host manages TCP/IP operations, thanks to resources provided by socket components dealing with procedures such as: connect(), send(), recv() and shutdown().

- The Proxy Client offers three high level procedures:
  - TLS-connect() realizes Phase I operations, either in full or resume mode.
  - TLS-write() encrypts and sends data in the Phase II.
  - TLS-read() reads and decrypts data in the Phase II.
Trafic Exchanges of Scard

```
[root@dhcp160-81 apps]# ./openssl s_scard -tlsv1 -debug -connect ubis03.enst.fr:443
CONNECTED (00000003)

SCARD: initializing smart card interface
Length of hexa dump = 16
SCARD: select file by AID : A0 00 00 00 30 00 02 FF FF FF FF 89 31 32 38 00

---
Acceptable client certificate CA names
---
SSL handshake has read 1927 bytes and written 1041 bytes
---
New, TLSv1/SSLv3, Cipher is RC4-MD5
Server public key is 1024 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
SSL-Session:
   Protocol : TLSv1
   Cipher : RC4-MD5
   Session-ID: 5A42DE4C101B6130D697D56AFA1AFD5B8A673CC8979DEFB38CBB0E46F1F9378A
   Session-ID-ctx:
   Master-Key: 0000000000000000000000000000000000000000000000000000000000000000
   Key-Arg : None
   PSK identity: None
   PSK identity hint: None
   SRP username: None
   Start Time: 1399479186
   Timeout : 7200 (sec)
Verify return code: 19 (self signed certificate in certificate chain)
```
**Trafic Exchanges of TLS Server**

```
[root@ubis03 apps]# openssl s_server -msg -debug -tssl -accept 443 -CAfile root.pem -verify 1 -cert servercert.pem -key serverkey.pem -pass pass:pascal

verify depth is 1
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT

Client certificate
-----BEGIN CERTIFICATE-----
MIICcTCCAdjoIBAgIBbzANBgkqhkiG9w0BAQUFADBBIjDELMAkGA1UEBhMCRIIX
DzANBgNVBAgTBkBZyYW5jZTEOMAwGA1UEBxMFUGFyaXMxEarBGbNVBAoTCKV0aGVu
VYHJ1c3QxDTALBgNVBAsTBFRlc3QxFDASBGbNVBATMCiBhc2NhbFVyaWVuMSowKAYJ
KoZIhvnaNCb8Fh5tYXnjWwwudXjZpZnSAZIRoXJZc5jZ02wHhMcNMTa5MDM2
MiA1OTlzW6kMNzjwMTAYMiA1OTlzW6kM0CAQDB0xZGQswCQYDVQQGEwFJU0Ew
GwYHMA0GCSqGSIb3DQEBAQUA4GNADCBiKBgQDDkzuOB1t/SURVv6ftAmfjmjwMBtvf
mmOVVztdsYG5Q2PTfKeK6cVIITGimZ0
/MsntviUQmFWkO5tG8RJKvNhFmbDCZt84YKdElfc4z/qzZEI0GdaMeuZ37jM0NQp
U6oyNB1R2INLQrc15ph83r7Vt48a40UHEjw0pCHfrQIDAQABowwCzAJBgNV
HRMEajAAMGCSqGSIb3DQEBAQUA4GGA1UdDQYJK4ECAQQDwI7a1UQLk0+XlJiyUR
07k2hY0C39e4QYy20xpn6jv5ViWxYkEcELnA6f8J sabeIhrOGeoB0f2jZi6EB
OGF7sLQ891iHQjB0o0NKe2jSMqg8Uw7h
8b/Dtnagd0HphXXa9gEnXwMgKnyQaGRbFN6re3ExCMOqh4q78c96npt3704
ZNiq7TmN98y
-----END CERTIFICATE-----

subject=/C=FR/ST=IleDeFrance/L=Paris/O=ethertrust.com/CN=client
issuer=/C=FR/ST=France/L=Paris/O=EtherTrust/OU=Test/CN=PascalUrien/emailAddress=pascal@enst.fr
Shared ciphers:RC4-MD5
CIPHER is RC4-MD5
Secure Renegotiation IS NOT supported
```
The hardware & software Design of SecFuNet
Remote APDU Call Secure protocol (RACS)

- Allows all the features needed for the remote use of secure elements.
  - Inventory of Grid of Secure Elements,
  - Information exchange with the secure elements,

- The RACS protocol works over the TCP transport layer and is secured by the TLS protocol.

- One of the main targets of the RACS protocol is to efficiently push a set of ISO 7816 requests towards a secure element in order to perform cryptographic operations.

- Typically, the RACS Protocol requests comprises:
  - a prefix made with multiple APDU ISO 7816 commands and
  - a suffix that collects the result of a cryptographic procedure.
Use case II: Interactions between User, VM & GoSE

TLS Connection Request

PHASE I
- [ClientHello]
- [ServerHello, Certificate]
  - [CertificateRequest, ServerHelloDone]
- [Certificate, CertificateVerify]
  - [ChangeCipherSpec, Finished]

PHASE II
- [ChangeCipherSpec, Finished]

TLS Encryption Data

RACS(ClientHello)
- RACS(ServerHello, Certificate)
- RACS(CertificateRequest, ServerHelloDone)
- RACS(Certificate, CertificateVerify, ChangeCipherSpec, Finished)
- RACS(ChangeCipherSpec, Finished)
- RACS(Get-CipherSuite, Get-KeyBlock)

EAP-Process (ClientHello)
- EAP-Response (ServerHello, Certificate, CertificateRequest, ServerHelloDone)
- EAP-Process (Certificate, CertificateVerify, ChangeCipherSpec, Finished)
- EAP-Response (ChangeCipherSpec, Finished)
- Get-CipherSuite
- Get-KeyBlock
Conclusion

• To integrate into OpenSSL, a some libraries dedicated to all smart cards.
• To trivialize and demystify the use of smart cards in development of sensitive applications,

• This kit (SDK-Smart-Card-API) provides to the developers:
  – an abstraction between applications and smart cards.
  – a simple primitives to manage these microcontrollers, especially our EAP-TLS smart card.
  – So, OpenSSL becomes compatible with PC/SC standard
REFERENCES & STANDARDS

[6]: P.Urien, ‘Remote APDU Call Secure », draft-urien-core-racs-00.txt, August 2013
[14]: OpenSc-Tools : https://www.opensc-project.org/opensc
[17]: Secfunet, "Infrastructure of the authentication server", Deliverable 2.1.
[18]: SecFunet, "Array and software of authentication server", Deliverable 2.2.
Thanks for your attention!

Have you any questions?

E-mail: hassane.aissaoui@telecom-paristech.fr