Smart Card support Embedded Within OpenSSL to Secure Virtual Machines

Authors :

Hassane AISSAOUI-MEHREZ — Télécom Paris Tech Institute
Pascal URIEN — Télécom Paris Tech Institute
Guy PUJOLLE — 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 ?

Introduction

- 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.

What about a Grid of Secure Elements (GoSE)?

- **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.**

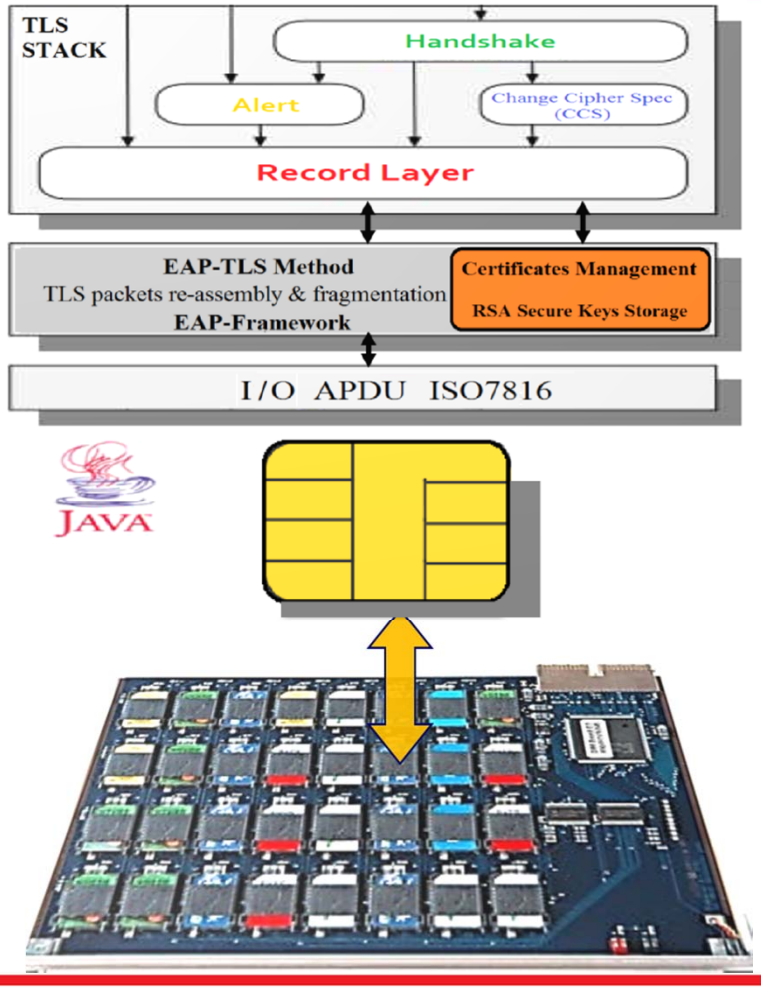
- Typically we can use it as a central SIM Server in distributed applications:
  - identifications of Virtual Machines,

  Or

  - isolation of Network on Demand (i.e. SDN).

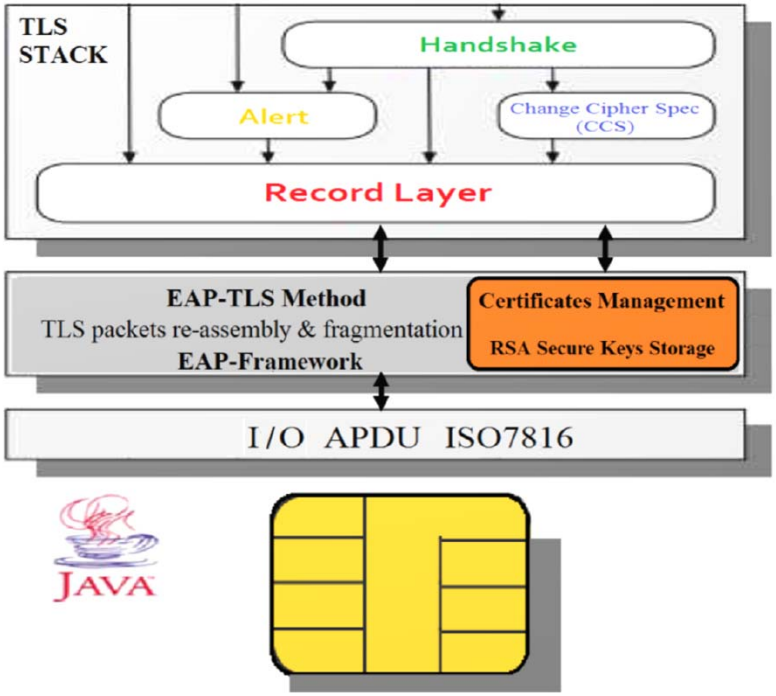
What about a Secure Element (SE)?

- 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.

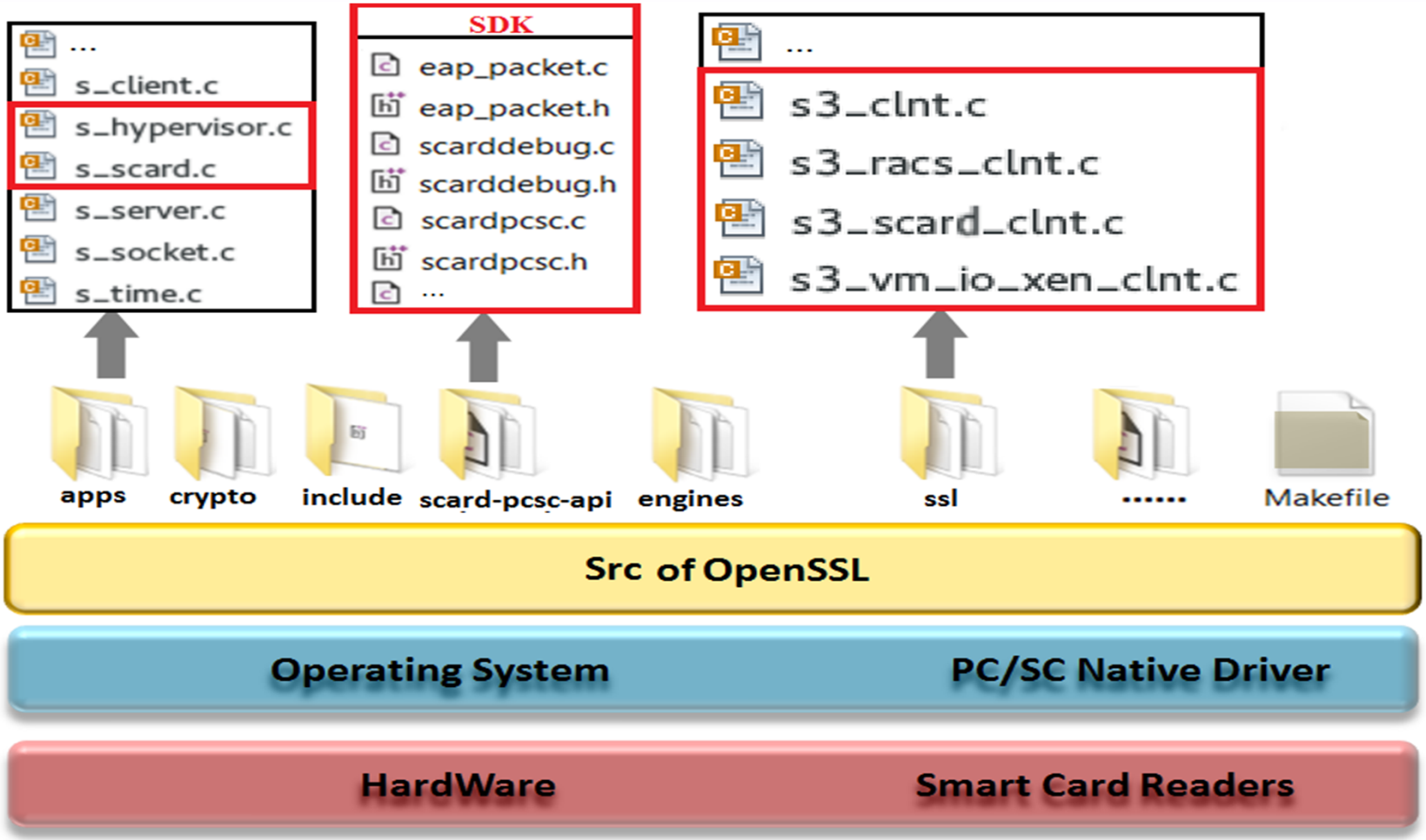


Software Design of SE

- **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

```
/*
 * smart card interface for EAP-TLS Smart Card
 * (/openssl/scard_pcsc_api/scardpcsc.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 {
    SCARDCONTEXT ctx;
    SCARDHANDLE card;
    DWORD protocol;
    sim_types sim_type;
    int pin1_required;
    unsigned long nbr_reader;
    unsigned char BufRecv[MAXTLSMSG];
    unsigned char BufSend[MAXTLSMSG];
    DWORD SendLen;
    DWORD RecvLen;
    struct wpabuf *CipherSuite;
    struct wpabuf *KeyBlock;
};
```

```
/*
 * Procedures for EAP-TLS Smart Card support
 *
 */
int scard_make_eaptls_msg_output(struct scard_data * scard,
                                 int fragments_size);
struct scard_data *scard_init_eaptls(scard_sim_type sim_type,
                                     const char *reader,
                                     int pin_needed,
                                     unsigned long readertohandle);
int scard_select_file_eaptls(struct scard_data *scard,
                             sim_types sim_type,
                             unsigned char *aid,
                             size_t aidlen);

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

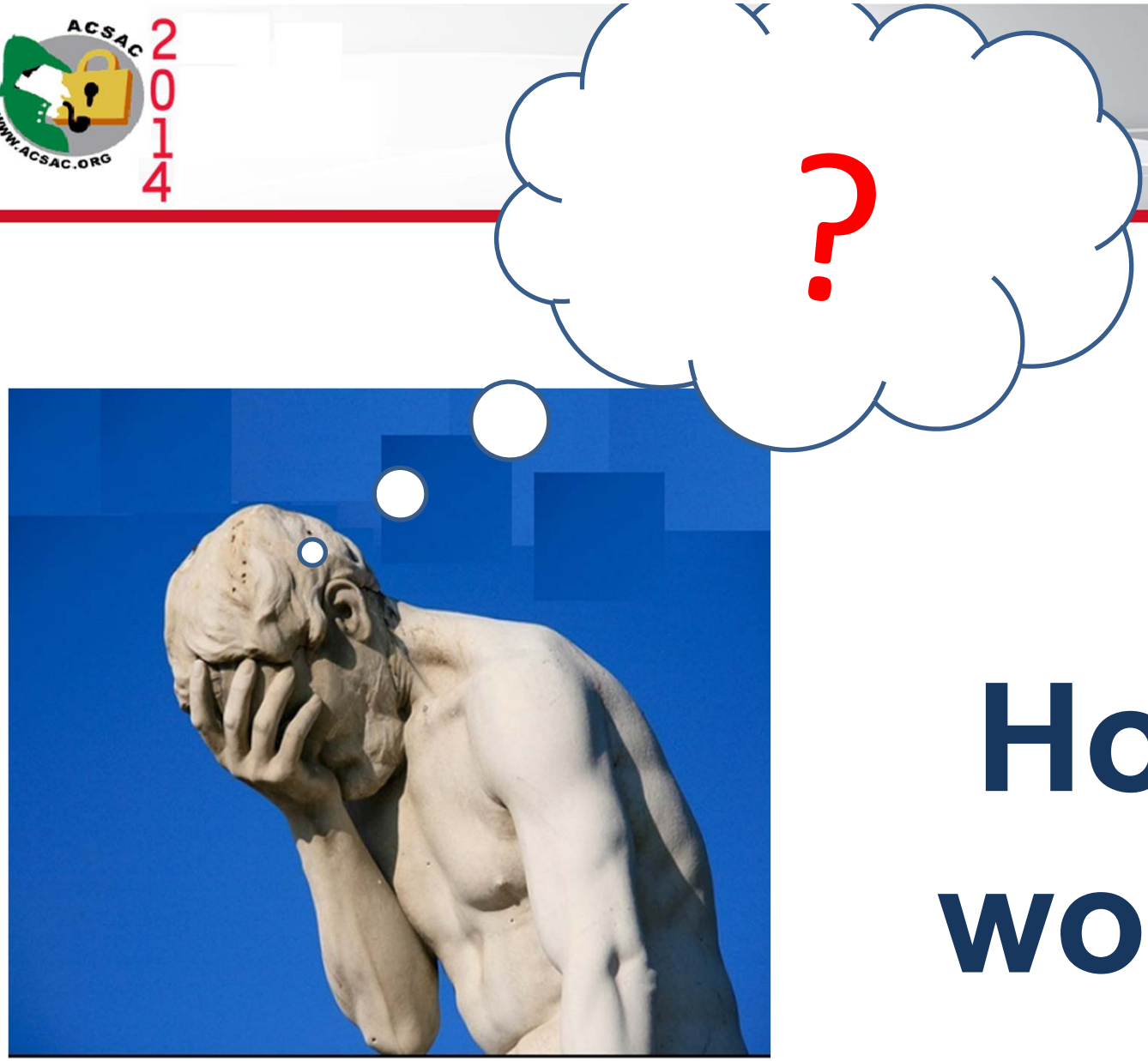
SDK-Smart-Card-API Product Offerings

Proxy Client, Proxy VM & Proxy Server Procedures

- scard_buffer_write(BIO* b, const char* in, int inl)
- scard_dump_callback(unsigned char* argdump, int arglen)
- scard_get_ciphersuite_keyblock(SSL* s, scard_data* smartcard)
- scard_get_client_key_exchange(SSL* s, scard_data* smartcard)
- scard_get_finished(SSL* s, int a, int b)
- scard_get_server_hello(SSL* s)
- scard_next_sock_write(BIO* b, unsigned char* in, int inl)
- scard_parse_certificate_request(SSL* s, unsigned char buf[], int init_num)
- scard_process_eaptls_request(scard_data* smartcard)
- scard_save_server_finished_resp(SSL* s, scard_data* smartcard)
- scard_save_serverhello_resp(SSL* s, scard_data* smartcard)
- scard_send_client_certificate(SSL* s, scard_data* smartcard)
- scard_send_client_hello(SSL* s, scard_data* smartcard)
- scard_set_change_cipher_spec(SSL* s, scard_data* smartcard, int a, int b)
- scard_set_finished(SSL* s, scard_data* smartcard, int a, int b, const char* sender)
- scard_set_sequence(SSL* s, int send)
- scard_setup_key_block(SSL* s, scard_data* smartcard)
- scard_write_pending(SSL* s, int type, const unsigned char* buf, unsigned int len)
- scard_write_sock(SSL* s, int type)
- remote_vmtoxen_usage()
- send_vmtoxen_cmd(SSL* ssl, char* cmd, int len)
- vm_to_xen_ssl3_connect(SSL* tls_vmtoxen_ctx, SSL* s)
- vmtoxen_buffer_write(BIO* b, const char* in, int inl)
- vmtoxen_dump_callback(unsigned char* argdump, int arglen)
- vmtoxen_get_ciphersuite_keyblock(SSL* tls_vmtoxen_ctx, SSL* s, char KeyBlock[])
- vmtoxen_get_client_key_exchange(SSL* tls_vmtoxen_ctx, SSL* s, char CipherSuite[])
- vmtoxen_get_finished(SSL* s, int a, int b)
- vmtoxen_get_server_hello(SSL* s)
- vmtoxen_next_sock_write(BIO* b, unsigned char* in, int inl)
- vmtoxen_parse_certificate_request(SSL* s, char buf[], int init_num)
- vmtoxen_process_client_certificate(SSL* tls_vmtoxen_ctx, SSL* s, char ClientCertRec)
- vmtoxen_process_eaptls_request(scard_data* smartcard)
- vmtoxen_save_server_finished_resp(SSL* s, char* SrvFinished)
- vmtoxen_save_serverhello_resp(SSL* s, char* SrvHello, int offset)
- vmtoxen_send_client_hello(SSL* tls_vmtoxen_ctx, SSL* s)
- vmtoxen_set_change_cipher_spec(SSL* s, char* BufRecv, int a, int b)
- vmtoxen_set_finished(SSL* s, char* BufRecv, int a, int b, const char* sender, int slen)
- vmtoxen_set_sequence(SSL* s, int send)
- vmtoxen_setup_key_block(SSL* s, char* KeyBlock, int KeyBlockLen)
- vmtoxen_ssl3_check_cert_and_algorithm(SSL* s)
- vmtoxen_ssl3_check_finished(SSL* s)
- vmtoxen_ssl3_client_hello(SSL* s)
- vmtoxen_ssl3_get_cert_status(SSL* s)
- vmtoxen_ssl3_get_certificate_request(SSL* s)
- vmtoxen_ssl3_get_key_exchange(SSL* s)
- vmtoxen_ssl3_get_new_session_ticket(SSL* s)
- vmtoxen_ssl3_get_server_certificate(SSL* s)
- vmtoxen_ssl3_get_server_done(SSL* s)
- vmtoxen_ssl3_get_server_hello(SSL* s)

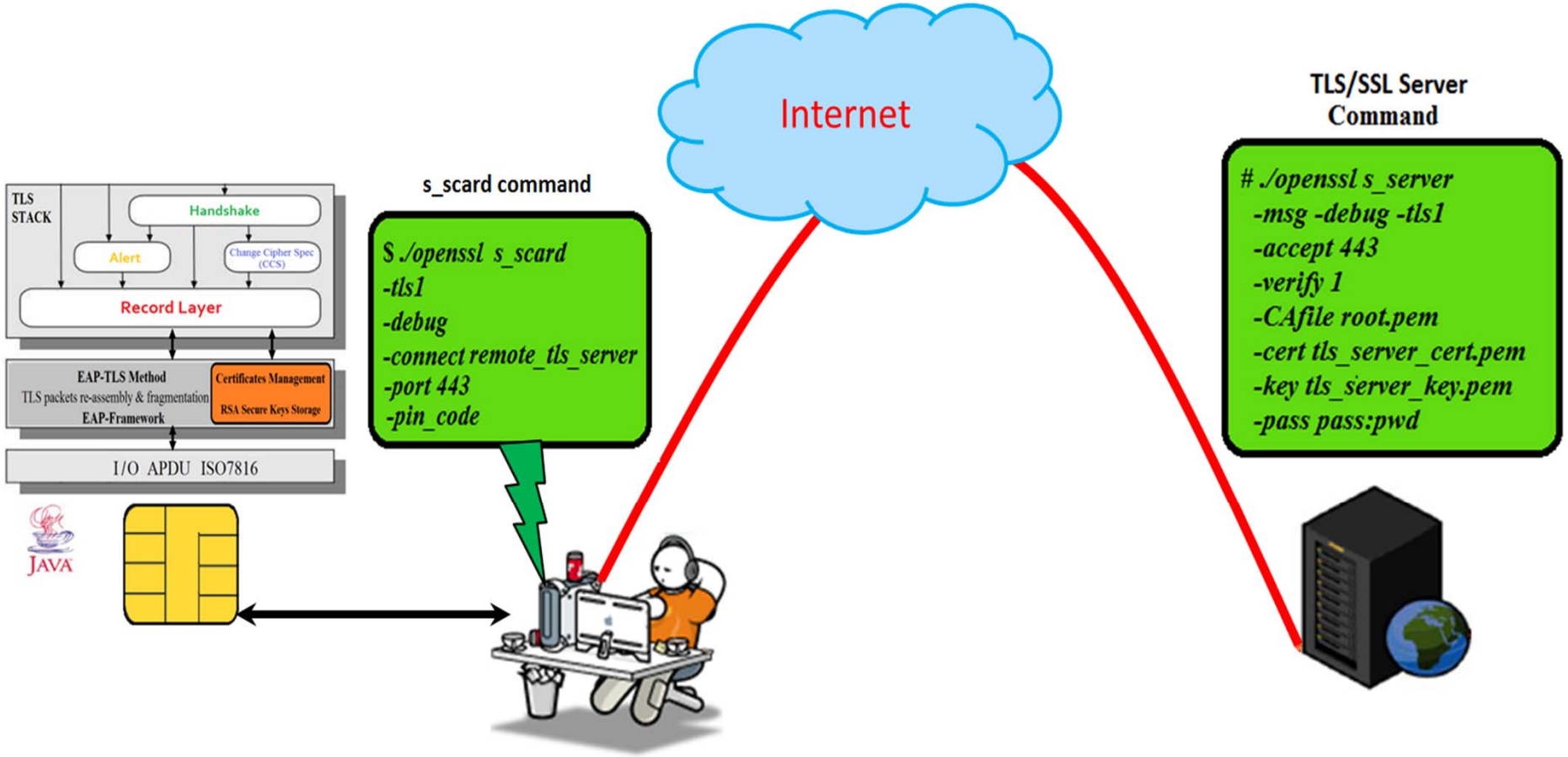
Proxy RACS Procedures

- racs_buffer_write(BIO* b, const char* in, int inl)
- racs_dump_callback(unsigned char* argdump, int arglen)
- racs_get_ciphersuite_keyblock(SSL* ssl_gose_ctx, SSL* s, char CipherSuite[])
- racs_get_client_key_exchange(SSL* ssl_gose_ctx, SSL* s, char CipherSuite[])
- racs_get_finished(SSL* s, int a, int b)
- racs_get_server_hello(SSL* s)
- racs_init_remote_scard(SSL* ssl)
- racs_next_sock_write(BIO* b, unsigned char* in, int inl)
- racs_parse_certificate_request(SSL* s, char buf[], int init_num)
- racs_process_client_certificate(SSL* ssl_gose_ctx, SSL* s, char ClientCertRecv[])
- racs_process_eaptls_request(scard_data* smartcard)
- racs_save_server_finished_resp(SSL* s, char* SrvFinished)
- racs_save_serverhello_resp(SSL* s, char* SrvHello, int offset)
- racs_send_client_hello(SSL* ssl_gose_ctx, SSL* s)
- racs_set_change_cipher_spec(SSL* s, char* BufRecv, int a, int b)
- racs_set_finished(SSL* s, char* BufRecv, int a, int b, const char* sender, int slen)
- racs_set_sequence(SSL* s, int send)
- racs_setup_key_block(SSL* s, char* KeyBlock, int KeyBlockLen)
- racs_ssl3_check_cert_and_algorithm(SSL* s)
- racs_ssl3_check_finished(SSL* s)
- racs_ssl3_client_hello(SSL* s)
- racs_ssl3_connect(SSL* ssl_gose_ctx, SSL* s)
- racs_ssl3_get_cert_status(SSL* s)
- racs_ssl3_get_certificate_request(SSL* s)
- racs_ssl3_get_key_exchange(SSL* s)
- racs_ssl3_get_new_session_ticket(SSL* s)
- racs_ssl3_get_server_certificate(SSL* s)
- racs_ssl3_get_server_done(SSL* s)
- racs_ssl3_get_server_hello(SSL* s)
- racs_ssl3_send_client_certificate(SSL* s)
- racs_ssl3_send_client_key_exchange(SSL* s)
- racs_ssl3_send_client_verify(SSL* s)
- racs_ssl3_send_next_proto(SSL* s)
- racs_ssl_do_client_cert_cb(SSL* s, X509** px509, EVP_PKEY** ppkey)
- racs_write_pending(SSL* s, int type, const unsigned char* buf, unsigned int len)
- racs_write_sock(SSL* s, int type)
- remote_fetch_msg_from_GoSE(SSL* ssl, char BufRecv[])
- remote_tls_usage()
- send_racs_cmd(SSL* ssl, char* cmd, int len)



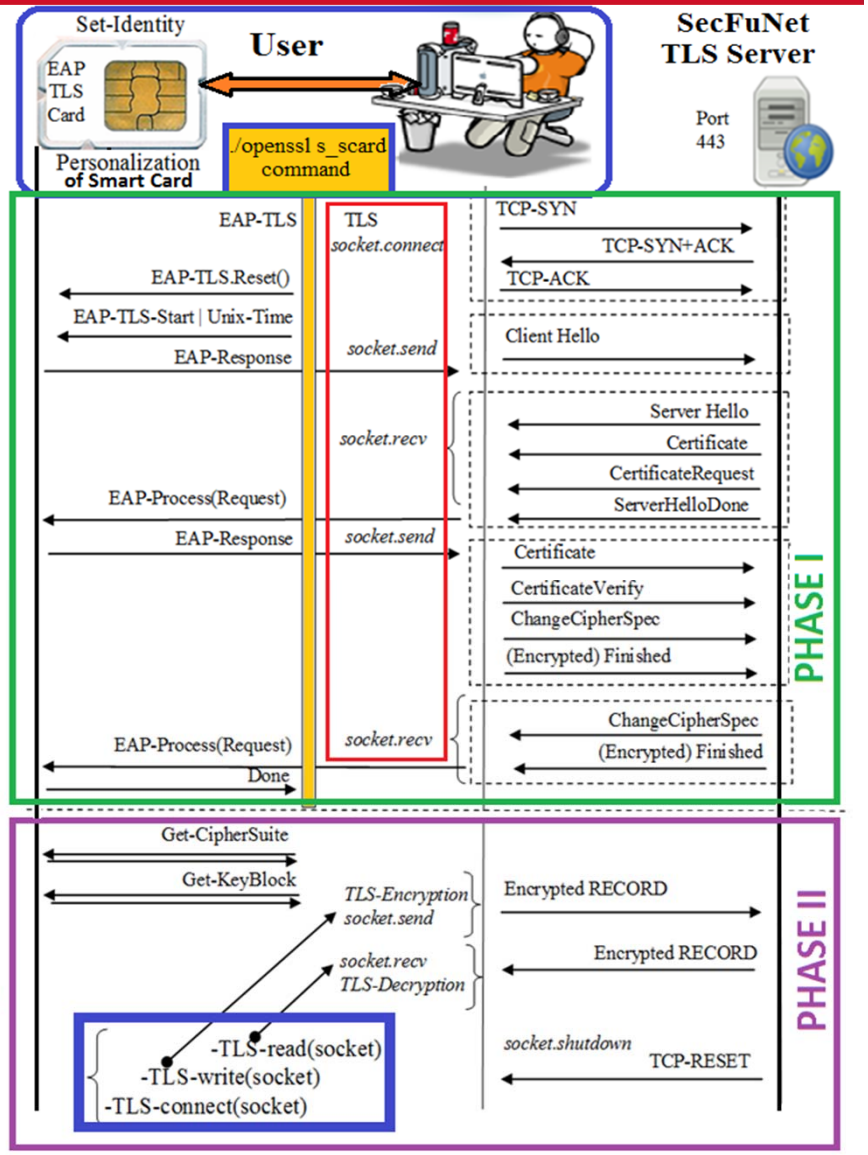
How it works?

How it works?


Internet
TLS STACK
Handshake
Alert
Change Cipher Spec (CCS)
Record Layer
EAP-TLS Method
TLS packets re-assembly & fragmentation
EAP-Framework
Certificates Management
RSA Secure Keys Storage
I/O APDU ISO7816
JAVA
s_scard command
$ ./openssl s_scard
-tls1
-debug
-connect remote_tls_server
-port 443
-pin_code
TLS/SSL Server Command
./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

```
[root@ubis03 apps]# openssl s_server -msg -debug -tls1-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-----
MIICdTCCAd6gAwIBAgIBbzANBgkqhkiG9w0BAQUFADCBlDELMAkGA1UEBhMCRlIx
DzANBgNVBAgTBkZyYW5jZTE0MAwGA1UEBxMFUGFyaXMxEzARBgNVBAoTCkV0aGVy
VHJ1c3QxDTALBgNVBAsTBFRlc3QxFDASBgNVBAMTC1Bhc2NhbFVyaWVuMSowKAYJ
KoZIhvcNAQkBFhtwYXNjYWwudXJpZW5AZXRoZXJ0cnVzdC5jb20wHhcNMTAxMDE2
MjA1OTEzWhcNMTkwMTAyMjA1OTEzWjBdMQswCQYDVQQGEwJGUjEUMBIGA1UECBML
SWxlRGVGcmFuY2UxDjAMBgNVBAcTBVBhcmlzMRcwFQYDVQQKEw5ldGhlcnRydXN0
LmNvbTEPMA0GA1UEAxMGY2xpZW50MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQDkzu0Bit/SUrRV6FtmApmFjmgZBVotFmmOvGZtdsYG5Q2PTfKeKGcViTGiMVZO
/MsnqUFQQmwKOtG8RJkVhFmbDCZt84YkRdEIfc4z/qzZEIGOgdaMeuZ37jMG0NQp
U6oyNbBP1R2NXlQrci5phR3rb7Vt48a4OUHEwj0pCHfjrQIDAQABow0wCzAJBgNV
HRMEAjAAMA0GCSqGSIb3DQEBBQUAA4GBAIT+0rWCW7WP+hU6s8L6hFOYFhV1kU0A
0XGw/GG2G5LOMzvnd2x4hPjic6bEZvKKuEwslQ891jIHjBOOoNKVeiSMGvq8Uw7h
8b/DtNagd0HpWXAwo9gENwXwMgKnyQaGRbFN0roe3ExmC0PqH4qf78c96npt37o4
ZNiqtTxN098y
-----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
```

```
[root@dhcp160-81 apps]# ./openssl s_scard  -tls1 -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 Scard

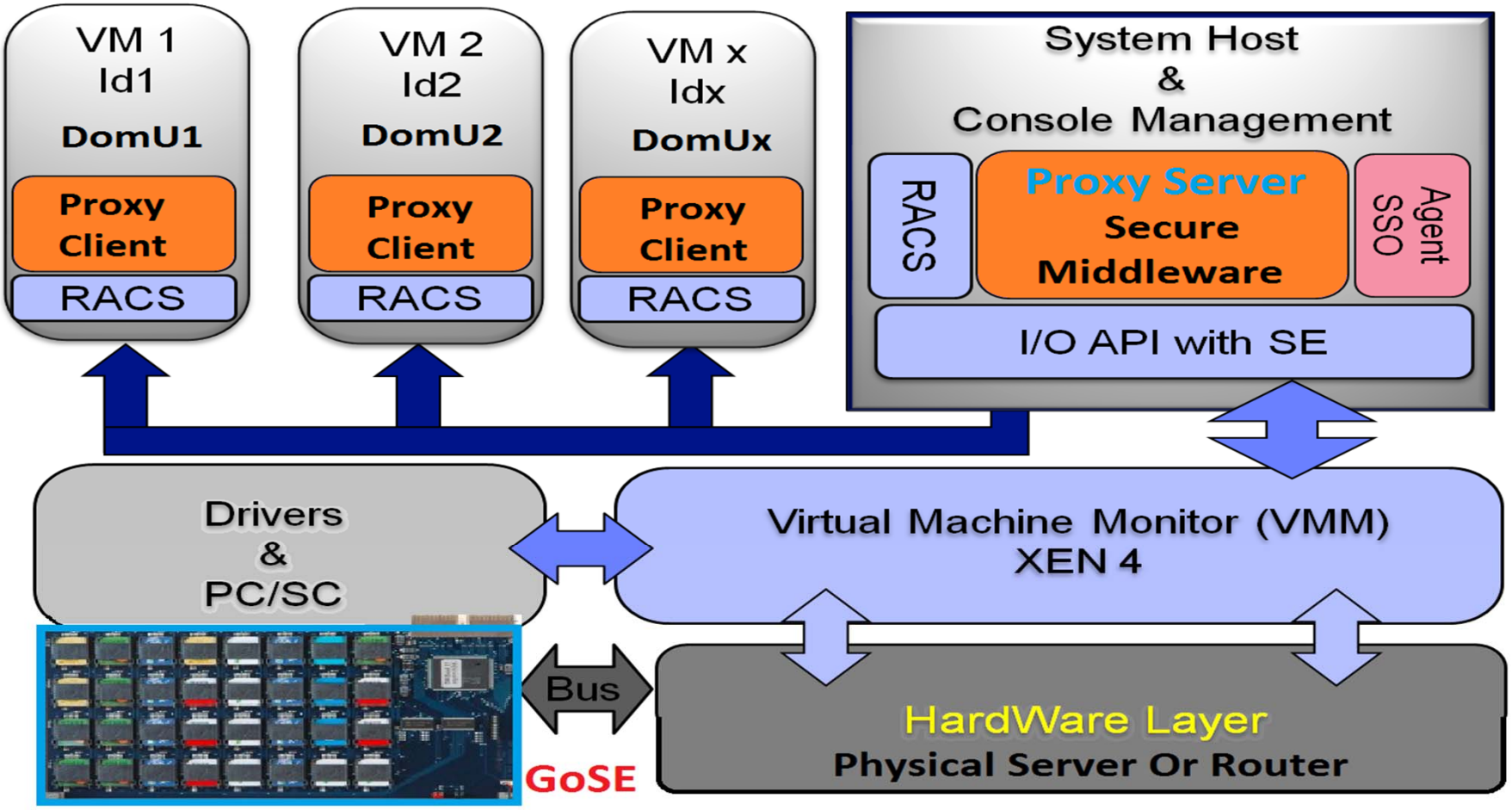
```
[root@dhcp160-81 apps]# ./openssl s_scard  -tls1 -debug -connect ubis03.enst.fr:443
CONNECTED(00000003)

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: 00000000000000000000000000000000000000000000000000000000000000
    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 -tls1-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-----
MIICdTCCAd6gAwIBAgIBbzANBgkqhkiG9w0BAQUFADCBlDELMAkGA1UEBhMCRlIx
DzANBgNVBAgTBkZyYW5jZTE0MAwGA1UEBxMFUGFyaXMxEzARBgNVBAoTCkV0aGVy
VHJ1c3QxDTALBgNVBAsTBFRlc3QxFDASBgNVBAMTC1Bhc2NhbFVyaWVuMSowKAYJ
KoZIhvcNAQkBFhtwYXNjYWwudXJpZW5AZXRoZXJ0cnVzdC5jb20wHhcNMTAxMDE2
MjA1OTEzWhcNMTkwMTAyMjA1OTEzWjBdMQswCQYDVQQGEwJGUjEUMBIGA1UECBML
SWxlRGVGcmFuY2UxDjAMBgNVBAcTBVBhcmlzMRcwFQYDVQQKEw5ldGhlcnRydXN0
LmNvbTEPMA0GA1UEAxMGY2xpZW50MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQDkzu0Bit/SUrRV6FtmApmFjmgZBVotFmm0vGZtdsYG5Q2PTfKeKGcViTGiMVZ0
/MsnqUFQQmwK0tG8RJkVhFmbDCZt84YkRdEIfc4z/qzZEIG0gdaMeuZ37jMG0NQp
U6oyNbBP1R2NXlQrci5phR3rb7Vt48a4OUHEwj0pCHfjrQIDAQABow0wCzAJBgNV
HRMEAjAAMA0GCSqGSIb3DQEBBQUAA4GBAIT+0rWCW7WP+hU6s8L6hF0YFhV1kU0A
0XGw/GG2G5L0Mzvnd2x4hPjic6bEZvKKuEwslQ891jIHjBO0oNKVeiSMGvq8Uw7h
8b/DtNagd0HpWXAwo9gENwXwMgKnyQaGRbFN0roe3ExmC0PqH4qf78c96npt37o4
ZNiqtTxN098y
-----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

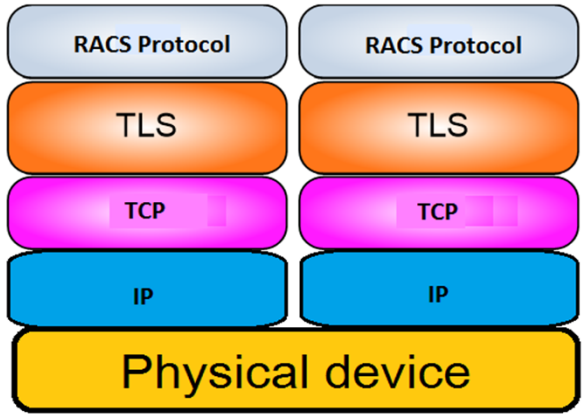


Remote APDU Call Secure protocol (RACS)

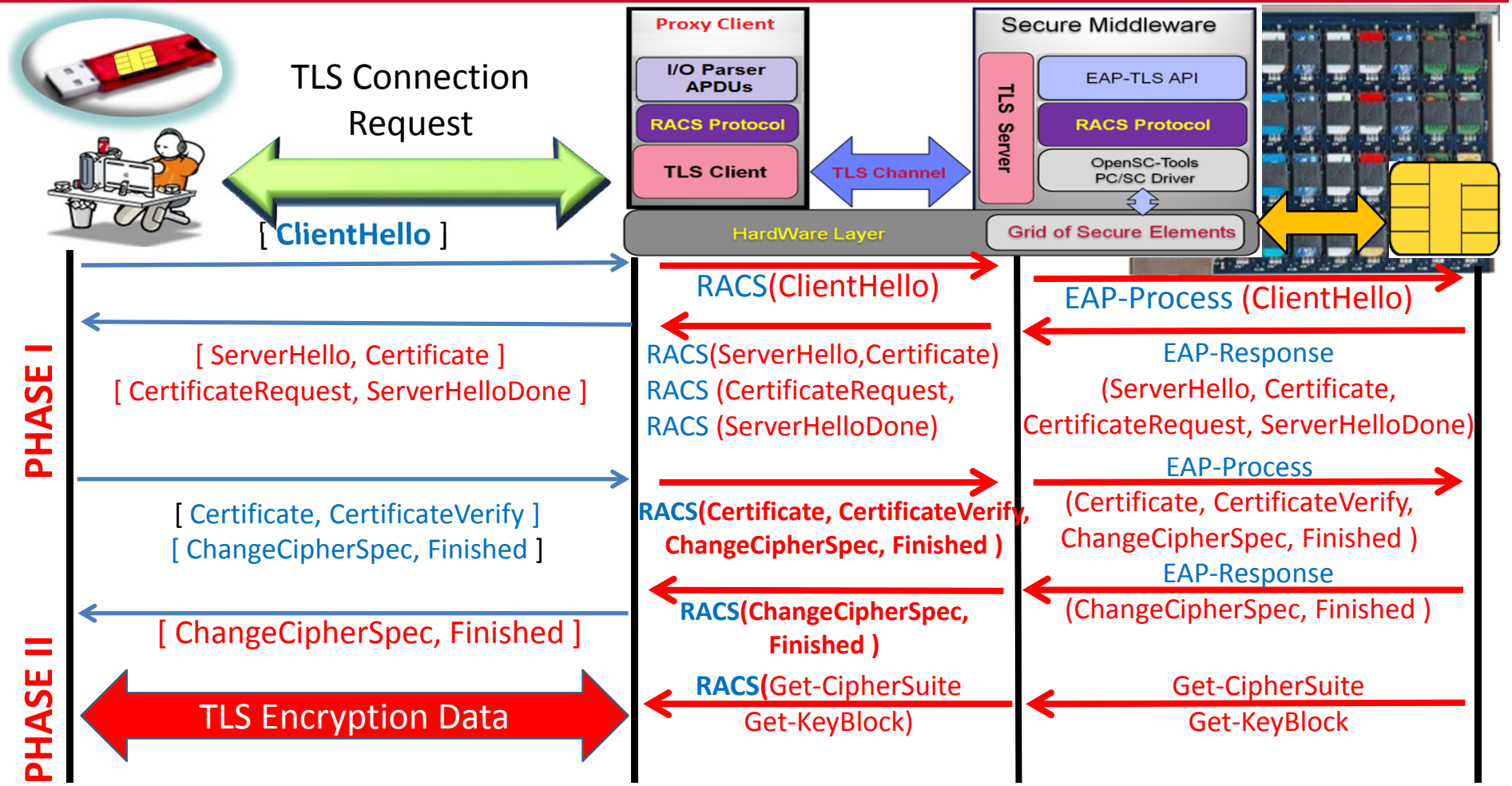
- Allows all the features needed for the remote use of secure elements.
  - Inventory of Grid of Secure Elements,
  - Information exchange with the secure elements,
- The RACS protocol works over the TCP transport layer and is secured by the TLS protocol.
- One of the main targets of the RACS protocol 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



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

[1]: RFC 3748, "Extensible Authentication Protocol, (EAP)", June 2004.

[2]: IETF draft, "EAP-Support in Smart card", draft-urien-eap-smartcard-25.txt, July 2013.

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

[4]: Chen, Z., "Java CardTM Technology for Smart cards: Architecture and Programmer's (The Java Series) ", Addison-Wesley Pub Co 2002, ISBN 020170329.

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

[6]: P.Urien, 'Remote APDU Call Secure », draft-urien-core-racs-00.txt, August 2013

[7]: N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," Comput.Netw., vol. 54, pp. 862–876, Apr. 2010.

[8]: J. Turner and D. Taylor, "Diversifying the internet," in Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE, vol. 2, pp. 6 pp.–760, 2005.

[9]: D. Clark, J. Wroclawski, K. Sollins, and R. Braden, "Tussle in cyberspace: defining tomorrow's internet," Networking, IEEE/ACM Transactions on, vol. 13, no. 3, pp. 462–475, 2005.

[10]: T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," Computer, vol. 38, no. 4, pp. 34–41, 2005.

[11]: N. Fernandes, M. Moreira, Moraes, et al., "Virtual networks: Isolation, performance, and trends," Annals of Telecomm., pp. 1–17, 2010.

[12]: Urien, P., "Cloud of Secure Elements, Perspectives for Mobile and Cloud Applications Security", First IEEE Conference on Communications and Network Security" IEEE CNS 2013, 16-19 october 2013, DC USA.

[13]: R. Couto, M. Campista, and L. H. M. K. Costa, "Xtc: A throughput control mechanism for xen-based virtualized software routers," in Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, pp. 1–6, 2011.

[14]: OpenSC-Tools : https://www.opensc-project.org/opensc

[15]: Urien, P., "Cloud of Secure Elements, Perspectives for Mobile and Cloud Applications Security", First IEEE Conference on Communications and Network Security" IEEE CNS 2013, 16-19 october 2013, DC USA.

[16]: PKCS#11, https://latinamerica.rsa.com/rsalabs/node.asp?id=2133.

[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