Rapport de Stage

Simple Certificate Validation Protocol

Etude du protocole et implémentation de la vérification dans OpenSSL

Yann BUSNEL
ENST Bretagne - Rennes
Le 31 Juillet 2003
Stage effectué sur la période Juin-Juillet 2003.

Table des matières

Préambule

J'ai suivi au cours de l'année 2002-2003, la première année du Magistère Informatique et Telecommunication au sein de l'Université Rennes 1 et de l'Ecole Normale Supérieure de Cachan - Antenne de Bretagne. Ce rapport reprend mes différents travaux au sein du Projet RSM, à l'ENST Bretagne, sur le Campus de Beaulieu à Rennes, durant un stage d'été de deux mois Juin-Juillet 2003.

Ce stage m'a été proposé par Francis Dupont, Chercheur à l'ENST Bretagne - Antenne de Rennes. Le sujet était l'étude de SCVP (Simple Certificate Validation Protocol [1]) avec comme objectif final l'implémentation de la vérification via SCVP dans OpenSSL (partie client : cf. Annexe C, page 25) et du service SCVP (partie serveur : cf. Annexe D, page 35).

Ce stage m'a permis de confirmer ma vocation d'enseignant-chercheur et d'observer le fonctionnement d'un autre laboratoire de recherche en informatique non affilié INRIA/CNRS.

Remerciements

Je tiens à remercier tout d'abord, Claude JARD et Luc BOUGÉ, chercheurs à l'INRIA/CNRS et Professeurs à l'ENS Cachan - Antenne de Bretagne, pour le stage qu'ils m'ont proposé d'accomplir.

Je remercie par ailleurs Francis DUPONT qui a été un maître de stage des plus instructifs et intéressants.

Merci à Géraldine TEXIER qui saura pourquoi, et à Gildas VALY pour ses dépannages précieux.

Un grand merci également à tout ceux et toutes celles qui m'ont aidé à rédiger et corriger ce rapport.

Chapitre 1

Certificats Numériques et PKI

Dans la cryptographie asymétrique, ou à clé privée, l'utilisation des bi-clés repose sur le postulat qu'à tout moment, l'une des parties de l'échange peut-être certaine de la validité du couple - Clé publique / Identité de l'utilisateur - de l'autre partie. Dans le cas contraire un tiers pourrait usurper l'identité de l'un des correspondants et ainsi utiliser sa signature.

Le certificat apporte une réponse à cette problématique. Un certificat de clé publique est une sorte de "carte d'identité numérique" qui atteste la validité de ce couple. Ce ne sont pas les seuls certificats existants (par exemple, les certificats d'attributs).

Plus précisément, un certificat de clé publique est une structure de données contenant une clé publique, des informations sur le propriétaire du certificat, l'ensemble étant signé par une autorité appelée Autorité de Certification (la CA : Certificate Authority).

1.1 Certificats X.509

Le format de certificat le plus utilisé dans le cadre d'une PKI (Public Key Infrastructure) suis la norme X.509 (version 3 - ISO IS 9594-8). Ce dernier permet l'utilisation des protocoles normalisés ou des applications tels que SSL, IPsec ou S/MIME.

La norme X.509v3 spécifie la nature des champs de la structure de données constituant le certificat. Le standard internet RFC 3280 décrit une déclinaison du format des certificats X.509 pour le monde de l'Internet.

Les entités utilisatrices de certificats correspondent aux différentes parties entrant en jeu dans le cadre d'un échange électronique : celles-ci peuvent donc être de différents types (utilisateurs, serveurs, applications, composants réseaux, ...)

1.2 Autorités de Certification

L'autorité de certification est le tiers de confiance dont la signature apparaît sur le certificat. Elle est responsable du processus de certification dans sa globalité.

Elle délègue parfois une partie de ses responsabilités à un opérateur de certification, chargé de la réalisation des opérations techniques liées à la certification. Dans ce cas l'Autorité de Certification assure un contrôle de l'opérateur de Certification.

La certification consiste à apposer la signature de l'Autorité de Certification sur le "gabarit" du certificat du demandeur. La CA possède une base de données interne dans laquelle elle stocke le statut de l'ensemble des certificats qu'elle a créé.

1.3 Public Key Infrastructure

L'Infrastructure à Clé Publique ou PKI désigne l'ensemble des moyens et ressources nécessaires à la gestion du cycle de vie et à l'exploitation des certificats.

1.3.1 Présentation

Une PKI regroupe ainsi les composants techniques, les ressources, l'organisation et les politiques de sécurité permettant l'usage de la cryptographie à clé publique dans un contexte défini.

Les fonctionnalités offertes par une PKI via ses différentes composantes permettent d'assurer la gestion du cycle de vie d'un certificat, dont un exemple est représenté en figure 1.1.

Une PKI s'articule généralement au moins autour des trois composants de base suivants :

- L'autorité de Certification qui crée et signe les certificats
- Une ou plusieurs Autorités d'Enregistrement (RA : Registration Authority) qui enregistrent et valident les demandes de certificats.
- Un dépôt qui stocke les certificats et les listes de certificats révoqués (CRL : Certificate Revocation List)

la figure 1.2 représente l'architecture type d'une PKI.

1.3.2 Révocation des certificats

La CA est donc responsable de la création du certificat. Elle doit également gérer l'ensemble du cycle de vie du certificat, notamment les fonctions de renouvellement, de révocation et de suspension.

Un certificat a une durée de vie limitée (entre un et trois ans pour un certificat utilisateur, et entre quatre et vingt ans pour le certificat d'une CA).

Des mécanismes doivent être prévus pour invalider un certificat avant sa date de fin de validité, c'est à dire pour le révoquer. Plusieurs causes peuvent

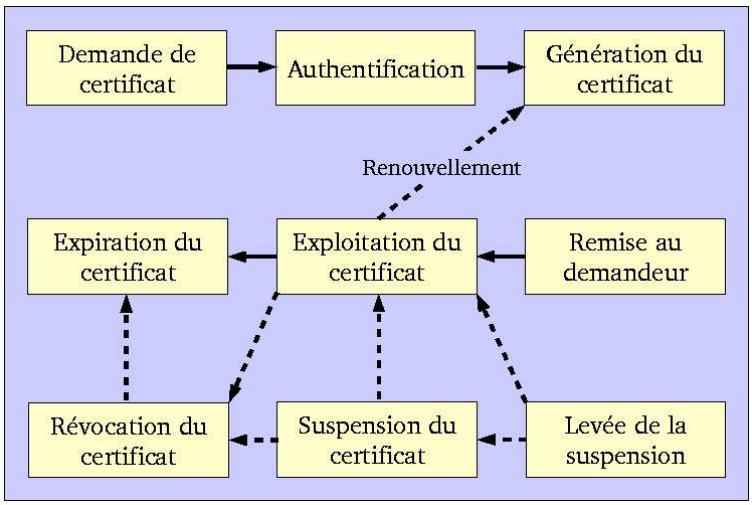

FIG. 1.1 – Cycle de vie d'un certificat

déclencher une révocation (compromission de la clé privée, informations obsolètes contenues dans le certificat, dysfonctionnement du support du certificat, ...)

La CA dispose de plusieurs méthodes pour répercuter la révocation d'un certificat :

- La révocation par annuaire positif : La CA retire de sa base chacun des certificats révoqués. Ainsi, les applications qui accèdent au dépôt n'ont accès qu'aux certificats valides.
- La CA publie des CRL qui sont des listes contenant les identifiants de chacun des certificats révoqués et non expirés. L'application récupère périodiquement une copie de la dernière CRL émise par la CA. Losqu'elle cherche à vérifier la validité d'un certificat, elle peut ainsi vérifier si le certificat en question est dans cette liste.

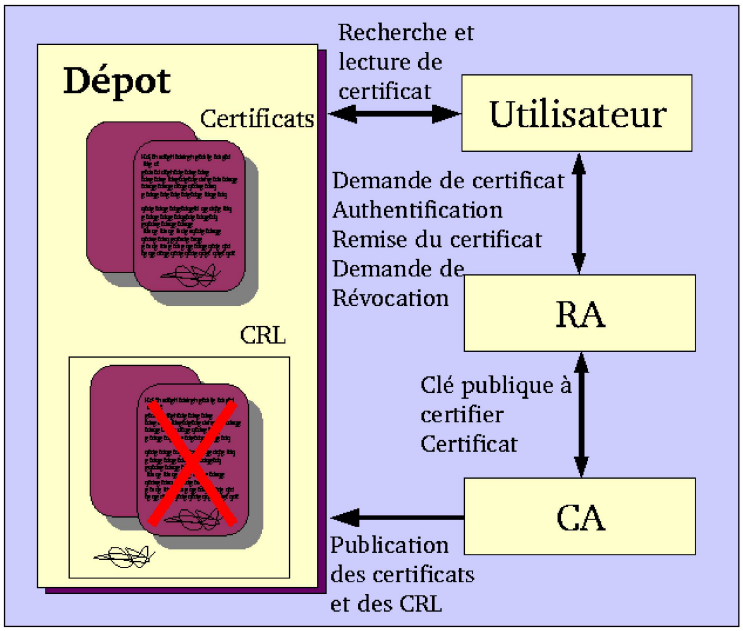

FIG. 1.2 – Structure d'une PKI

Chapitre 2

Définition de SCVP

Comme son nom l'indique, SCVP, ou Simple Certificate Validation Protocol, est un protocole de validation de certificat. Ce protocole est défini très précisément dans le draft internet - draft-ietf-pkix-scvp-12.txt [1]. Ci-après, le définition donnée sera plus succincte que ce document et orientée vers le cadre des travaux effectués durant ce stage.

2.1 Introduction

SCVP permet à un client de vérifier l'authenticité et la révocation d'un certificat sans posséder lui-même une PKI. Ce client envoie une requête au serveur SCVP qui s'occupe de remonter la chaîne de certificat.

Pour pouvoir valider un certificat, il faut demander à la CA si ce certificat est bien valable et non-révoqué. Mais, le plus souvent, cette CA est en fait une CA interne qui doit elle-même être certifiée par une CA plus importante. Ce principe de chaînage de certificat permet à des organismes d'avoir leur propre autorité de certification. A l'extrémité de ces chaînes de certificats se trouve des institutions telles que *VeriSign*.

La réponse du serveur SCVP au client peut contenir beaucoup d'informations. Notamment, il est possible de lui demander la validité d'un ou plusieurs certificats, le chemin d'accès permettant de vérifier la validité du certificat, le statut de celui-ci (révoqué, ...).

SCVP permet donc de simplifier la vérification de certificat pour de petites applications par exemple. Il permet également aux compagnies de centraliser les informations sur les politiques de sécurité et les certificats.

Ceci évite d'alourdir inutilement une application. Ce n'est pas elle qui se doit de faire les vérifications, elle peut donc continuer de tourner en attendant la réponse du serveur SCVP.

2.2 Requête

Comme expliqué plus haut, SCVP fonctionne sur le principe simple du modèle Requête-Réponse. Dans notre cas, chacun des messages echangés est encapsulé dans un message MIME de type `application/scvp-request` selon le protocole HTTP.

Il existe deux formes de requêtes : signée ou non-signée. La version signée permet d'authentifier le client au serveur. Mais, la structure de la requête reste la même.

Beaucoup d'informations peuvent être demandées au serveur mais l'énumération ci-dessous ne reprends que celles utilisées par la suite.

La requête est donc composée de :

- La version du protocole utilisé (actuellement, seule la version 1 existe) ;
- Le ou les certificats à valider ;
- La demande de vérification (validation simple, recherche du chemin d'accès, ...) ;
- La demande de réponse (le chemin d'accès, le statut, les informations sur la clé publique, ...) ;
- Le nom du demandeur permettant d'identifier le client (souvent le nom de la machine émettrice avec le nom de domaine de la machine) ;
- Un code permettant de différencier les différentes requêtes en provenance d'un même client.

Dans notre cas, ce code est un nombre déterminé de manière pseudo-aléatoire sur 4 octets.

2.3 Réponse

La réponse, encapsulée dans un MIME de type `application/scvp-response`, possède une structure proche de la requête :

- La version du protocole utilisé ;
- La date de production de la réponse ;
- Le statut de la réponse (permettant de savoir si la requête ne possédait pas d'erreur et a bien été gérée par le serveur SCVP) ;
- La référence de la requête (qui est dans notre cas un hachage de la requête elle même, ceci permettant de vérifier qu'un tiers n'a pas modifié la demande) ;
- Le nom du demandeur tel qu'il existe dans la requête ;
- Le nom du serveur répondant, dans le même format que celui du demandeur ;
- Le certificat à valider ;
- Le statut du certificat (si il est bien formé, si les clés correspondent bien, ...) ;
- La date de validité des informations contenues dans la réponse du serveur SCVP ;
- La validité du certificat (Correct, Révoqué, Inconnu [2] ou Introuvable) ;

- La réponse demandée dans la requête (le chemin d'accès, le statut, les informations sur la clé publique, ...) ;
- La politique de sécurité (par défaut dans notre cas) ;
- Le code permettant d'associer la réponse à la requête.

Chapitre 3

Implémentation de SCVP

Dans le cadre du projet de recherche RSM à l'ENST Bretagne, Francis Dupont m'a demandé d'implémenter le protocole SCVP dans OpenSSL. Pour cela, un client et un serveur ont été conçus en C et appelés de différentes façons selon l'organisation suivante.

3.1 Organisation

L'architecture du client se présente de la facon suivante :
- L'utilisateur fournit au client un certificat et/ou un URL à vérifier.
- Le client construit une requête SCVP à partir des données de l'utilisateur, au format ASN.1 voulu.
- Puis, le client appelle la bibliothèque ASN.1 d'OpenSSL afin d'encoder la requête en DER.
- Enfin, il envoie le message encapsulé dans un MIME de type `application/scvp-request` à un serveur HTTP via CURL.
- Le client attend la réponse du serveur SCVP.
- Il décode alors le code ASN.1 reçu, recréant alors la structure ASN.1 de la réponse.
- Pour finir, le client renvoie la validité du certificat contenue dans la réponse.

Il est évident que le travail de vérification se fait entièrement par le serveur. Celui-ci se décompose comme suit :
- Un CGI-Bin reçoit une requête SCVP et appelle le programme de vérification.
- Ce programme déchiffre le code ASN.1 reçu afin d'en extraire le certificat à vérifier et les actions à effectuer.
- Puis il appelle la fonction `X509_verify_cert()` de la bibliothèque `crypto` d'OpenSSL.
- En fonction de la réponse de `certverify()` et des demandes du client, le

programme construit la réponse SCVP correspondante.
- Puis il encode la structure et la renvoie au client.

Pour implémenter ce couple client/serveur, on a besoin d'un certain nombre d'outils qui sont décrits par la suite.

3.2 Bibliothèque ASN.1 dans OpenSSL

Dans le draft IETF, les structures ASN.1 des requêtes/réponses sont décrites très précisément. Il a donc fallu créer ces structures afin que les deux programmes décrits plus haut puissent les manipuler.

Pour cela, il existe une bibliothèque `crypto/asn1` dans OpenSSL qui par l'intermédiaire de macros, construit les fonctions de création/destruction et de codage/décodage par expansion. Il m'a donc fallu comprendre le fonctionnement de ces macros, écrire le code de création automatique et recompiler OpenSSL afin d'y intégrer la bibliothèque SCVP.

La création se base sur deux fichiers : `scvp.h` qui contient la définition des types ASN.1 définis dans SCVP, et `scvp_asn.c` qui appelle les macros permettant de créer les structures ASN.1 et de les implémenter dans OpenSSL.

Par exemple, la structure générale de la requête se définit par :

```
/*  CVRequest ::= SEQUENCE {
 *    scvpVersion           INTEGER,
 *    query                 Query,
 *    requestor         [0] OCTET STRING OPTIONAL,
 *    requestNonce      [1] OCTET STRING OPTIONAL,
 *    reqExtensions     [2] Extensions OPTIONAL }
 */
typedef struct scvp_request_st {
  ASN1_INTEGER *scvpVersion;
  SCVP_QUERY *query;
  ASN1_OCTET_STRING *requestor;        /* OPTIONAL */
  ASN1_OCTET_STRING *requestNonce;     /* OPTIONAL */
  STACK_OF(X509_EXTENSION) *reqExtensions;
} SCVP_REQUEST;
```

et l'implémentation s'appelle par :

```
ASN1_SEQUENCE(SCVP_REQUEST) = {
  ASN1_SIMPLE(SCVP_REQUEST, scvpVersion, ASN1_INTEGER),
  ASN1_SIMPLE(SCVP_REQUEST, query, SCVP_QUERY),
  ASN1_IMP_OPT(SCVP_REQUEST, requestor, ASN1_OCTET_STRING, 0),
  ASN1_IMP_OPT(SCVP_REQUEST, requestNonce, ASN1_OCTET_STRING, 1),
  ASN1_IMP_SEQUENCE_OF_OPT(SCVP_REQUEST, reqExtensions, X509_EXTENSION, 2)
} ASN1_SEQUENCE_END(SCVP_REQUEST)
```

Cette macro construit les tables de description qui seront interprétées par les fonctions de manipulation.

Et `IMPLEMENT_ASN1_FUNCTIONS(SCVP_REQUEST)` déclare les fonctions.

3.3 Bibliothèque CURL pour HTTP

Une autre bibliothèque importante pour notre implémentation de SCVP est la `libcurl`. Celle-ci permet de gérer les transferts via HTTP (ainsi que la plupart des protocoles des URL). Nous avons donc dû étudier les différentes fonctions C fournies par la bibliothèque.

L'utilisation de celle-ci est très bien illustrée par l'exemple suivant. Pour envoyer une requête SCVP, il faut :

- initialiser un contexte curl avec la fonction `curl_easy_init()` ;
- fournir l'URL du destinataire (ici, l'adresse du CGI) avec `CURLOPT_URL` ;
- joindre le message et la taille de celui-ci avec `CURLOPT_POSTFIELDS` et `CURLOPT_POSTFIELDSIZE` ;
- ajouter le type MIME du message avec `CURLOPT_HTTPHEADER` ;
- enfin, paramétrer les options d'envoi telles que le tampon d'erreur, la fonction d'écriture, ...

Ce qui donne en C la suite de commande suivante :

```
curl = curl_easy_init();

curl_easy_setopt(curl, CURLOPT_URL, url)
curl_easy_setopt(curl, CURLOPT_POSTFIELDS,
                     (char *) sbuf)
curl_easy_setopt(curl, CURLOPT_POSTFIELDSIZE, len)
hlist = curl_slist_append(hlist,
      "Content-Type: application/scvp-request");
curl_easy_setopt(curl, CURLOPT_HTTPHEADER, hlist)
curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,
                     &WriteMemory_cb)
curl_easy_setopt(curl, CURLOPT_FILE, (void *) &chunk)
curl_easy_setopt(curl, CURLOPT_ERRORBUFFER, ebuf)
curl_easy_perform(curl)
```

3.4 Implémentation de l'API Client/Serveur

Une fois l'étude des différentes bibliothèques et macros utiles au développement de l'application terminée, nous avons commencé l'implémentation proprement dite de l'interface client/serveur SCVP. J'ai été particulièrement aidé par Francis Dupont au cours de cette phase de développement. Sa grande expérience a été un atout considérable dans l'efficacité du codage.

Le code de ces deux programmes est donné en annexe de ce rapport (client : Annexe C, serveur : Annexe D). Les premiers résultats sur des tests effectués en local sont plutôt concluants. Il reste encore de nombreuses améliorations à apporter au code de ces logiciels mais le protocole est en état de marche.

A mon départ, il ne restait plus qu'à coder le CGI-Bin réceptionnant les requêtes et appelant le programme du serveur SCVP. En réalité, l'équipe a opté pour passer directement les messages en HTTP.

Conclusion

L'objet de ce stage de fin de première année du MIT a consisté en l'étude du protocole SCVP et à l'implémentation de celui-ci dans la vérification de certificat d'OpenSSL. Francis Dupont m'a assisté tout au long de mon stage, et m'a permis de terminer à temps la totalité du programme de stage que nous avions élaborer ensemble courant avril.

Ce stage à l'ENST fut une expérience d'autant plus enrichissante par son sujet et sa localisation. En effet, j'ai pu largement développer mes connaissances sur les télécommunications et appréhender certains problèmes que peut engendrer la spécification d'un nouveau protocole. De plus, le fait de travailler dans un laboratoire indépendant des deux grands organismes que sont le CNRS et l'INRIA m'a permis d'avoir une vision différente de la recherche en informatique en France (Ce laboratoire appartient au GET).

Ce stage m'a conforté dans mon choix de carrière au sein de la recherche et m'a permis de m'ouvrir davantage sur ce milieu intéressant.

Annexe A

Définitions / Acronymes

API	Application Program Interface
ASN.1	Abstract Syntax Notation One
CA	Certificate Authority
CRL	Certificate Revocation List
CURL	Client URL Request Library
DER	Distinguished Encoding Rules
HTTP	HyperText Transfer Protocol
IETF	Internet Engeenering Task Force
IPsec	Internet Protocol Security
MIME	Multi-Purpose Internet Mail Extension
MIT	Magistère Informatique et Télécommunication
OCSP	Online Certificate Status Protocol
PKI	Public Key Infrastructure
PKIX	Internet X.509 Public Key Infrastructure

RA	Registration Authority
RFC	Request For Comments
S/MIME	Secure Multi-Purpose Internet Mail Extension
SCVP	Simple Certificate Validation Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
URL	Uniform Resource Locator
X.509	Public-key and attribute certificate frameworks

Annexe B

Bibliothèque ASN.1 pour SCVP

Ci-dessous, la bibliothèque que j'ai implémentée dans OpenSSL pour gérer les structures ASN.1 de SCVP. On peut voir qu'elle utilise le fichier de définition `openssl/scvp.h` que je n'ai pas joint en annexe en raison de sa trop grande taille.

```
#include <openssl/asn1.h>
#include <openssl/asn1t.h>
#include <openssl/scvp.h>

ASN1_CHOICE(SCVP_REVOCATIONINFO) = {
  ASN1_IMP_SEQUENCE_OF(SCVP_REVOCATIONINFO, value.crl, X509, 0),
  ASN1_IMP_SEQUENCE_OF(SCVP_REVOCATIONINFO, value.delta_crl, X509, 1),
  ASN1_IMP(SCVP_REVOCATIONINFO, value.ocsp, OCSP_RESPONSE, 2),
  /*  ASN1_IMP(SCVP_REVOCATIONINFO, value.other, SCVP_OTHERREVINFO, 3) */
} ASN1_CHOICE_END(SCVP_REVOCATIONINFO)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_REVOCATIONINFO)

ASN1_SEQUENCE(SCVP_REVOCATIONINFOS) = {
  ASN1_SEQUENCE_OF(SCVP_REVOCATIONINFOS, revocation_info, SCVP_REVOCATIONINFO)
} ASN1_SEQUENCE_END(SCVP_REVOCATIONINFOS)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_REVOCATIONINFOS)

ASN1_SEQUENCE(SCVP_TRUSTANCHOR) = {
  ASN1_SIMPLE(SCVP_TRUSTANCHOR, anchor, SCVP_PKCREF),
  ASN1_IMP_SEQUENCE_OF_OPT(SCVP_TRUSTANCHOR, certPolicies, ASN1_OBJECT, 0)
} ASN1_SEQUENCE_END(SCVP_TRUSTANCHOR)
```

```
IMPLEMENT_ASN1_FUNCTIONS(SCVP_TRUSTANCHOR)

ASN1_SEQUENCE(SCVP_VALIDPOLICY) = {
  ASN1_SIMPLE(SCVP_VALIDPOLICY, valPolicyId, ASN1_OBJECT),
  /* ASN1_IMP_OPT(SCVP_VALIDPOLICY, parameters, ASN1_ADB, 0) */
} ASN1_SEQUENCE_END(SCVP_VALIDPOLICY)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_VALIDPOLICY)

ASN1_SEQUENCE(ESS_ISSUERSERIAL) = {
  ASN1_SIMPLE(ESS_ISSUERSERIAL, issuer, GENERAL_NAME),
  ASN1_SIMPLE(ESS_ISSUERSERIAL, serialNumber, ASN1_INTEGER)
} ASN1_SEQUENCE_END(ESS_ISSUERSERIAL)

IMPLEMENT_ASN1_FUNCTIONS(ESS_ISSUERSERIAL)

ASN1_SEQUENCE(ESS_CERTID) = {
  ASN1_SIMPLE(ESS_CERTID, certHash, ASN1_OCTET_STRING),
  ASN1_OPT(ESS_CERTID, issuerSerial, ESS_ISSUERSERIAL)
} ASN1_SEQUENCE_END(ESS_CERTID)

IMPLEMENT_ASN1_FUNCTIONS(ESS_CERTID)

ASN1_SEQUENCE(SCVP_ATTRIBUTECERT) = {
} ASN1_SEQUENCE_END(SCVP_ATTRIBUTECERT)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_ATTRIBUTECERT)

ASN1_CHOICE(SCVP_PKCREF) = {
  ASN1_IMP(SCVP_PKCREF, value.cert, X509, 1),
  ASN1_IMP(SCVP_PKCREF, value.pkcRef, ESS_CERTID, 2)
} ASN1_CHOICE_END(SCVP_PKCREF)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_PKCREF)

ASN1_CHOICE(SCVP_ACREF) = {
  ASN1_IMP(SCVP_ACREF, value.attrCert, SCVP_ATTRIBUTECERT, 3),
  ASN1_IMP(SCVP_ACREF, value.acRef, ESS_CERTID, 4)
} ASN1_CHOICE_END(SCVP_ACREF)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_ACREF)

ASN1_CHOICE(SCVP_CERTREF) = {
  ASN1_IMP(SCVP_CERTREF, value.pkc, SCVP_PKCREF, 0),
  ASN1_IMP(SCVP_CERTREF, value.ac, SCVP_ACREF, 1)
} ASN1_CHOICE_END(SCVP_CERTREF)
```

```
IMPLEMENT_ASN1_FUNCTIONS(SCVP_CERTREF)

ASN1_SEQUENCE(SCVP_QUERY) = {
  ASN1_SEQUENCE_OF(SCVP_QUERY, queriedCerts, SCVP_CERTREF),
  ASN1_SEQUENCE_OF(SCVP_QUERY, checks, ASN1_OBJECT),
  ASN1_SEQUENCE_OF(SCVP_QUERY, wantBack, ASN1_OBJECT),
  ASN1_IMP_OPT(SCVP_QUERY, serverContextInfo, ASN1_OCTET_STRING, 0),
  ASN1_IMP(SCVP_QUERY, valPolicy, SCVP_VALIDPOLICY, 1),
  ASN1_IMP_OPT(SCVP_QUERY, validityTime, ASN1_GENERALIZEDTIME, 2),
  ASN1_IMP_SEQUENCE_OF_OPT(SCVP_QUERY, trustAnchors, SCVP_TRUSTANCHOR, 3),
  ASN1_IMP_SEQUENCE_OF_OPT(SCVP_QUERY, intermediateCerts, SCVP_PKCREF, 4),
  ASN1_IMP_OPT(SCVP_QUERY, revInfos, SCVP_REVOCATIONINFOS, 5),
  ASN1_IMP_SEQUENCE_OF_OPT(SCVP_QUERY, queryExtensions, X509_EXTENSION, 6)
} ASN1_SEQUENCE_END(SCVP_QUERY)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_QUERY)

ASN1_SEQUENCE(SCVP_REQUEST) = {
  ASN1_SIMPLE(SCVP_REQUEST, scvpVersion, ASN1_INTEGER),
  ASN1_SIMPLE(SCVP_REQUEST, query, SCVP_QUERY),
  ASN1_IMP_OPT(SCVP_REQUEST, requestor, ASN1_OCTET_STRING, 0),
  ASN1_IMP_OPT(SCVP_REQUEST, requestNonce, ASN1_OCTET_STRING, 1),
  ASN1_IMP_SEQUENCE_OF_OPT(SCVP_REQUEST, reqExtensions, X509_EXTENSION, 2)
} ASN1_SEQUENCE_END(SCVP_REQUEST)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_REQUEST)

ASN1_SEQUENCE(SCVP_REPLYWANTBACK) = {
  ASN1_SIMPLE(SCVP_REPLYWANTBACK, wb, ASN1_OBJECT),
  ASN1_SIMPLE(SCVP_REPLYWANTBACK, value, ASN1_OCTET_STRING)
} ASN1_SEQUENCE_END(SCVP_REPLYWANTBACK)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_REPLYWANTBACK)

ASN1_SEQUENCE(SCVP_REPLYCHECK) = {
  ASN1_SIMPLE(SCVP_REPLYCHECK, check, ASN1_OBJECT),
  ASN1_SIMPLE(SCVP_REPLYCHECK, status, ASN1_INTEGER)
} ASN1_SEQUENCE_END(SCVP_REPLYCHECK)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_REPLYCHECK)

ASN1_SEQUENCE(SCVP_CERTREPLY) = {
  ASN1_SIMPLE(SCVP_CERTREPLY, cert, SCVP_CERTREF),
  ASN1_SIMPLE(SCVP_CERTREPLY, replyStatus, ASN1_ENUMERATED),
  ASN1_SIMPLE(SCVP_CERTREPLY, replyValTime, ASN1_GENERALIZEDTIME),
```

```
  ASN1_SEQUENCE_OF(SCVP_CERTREPLY, replyChecks, SCVP_REPLYCHECK),
  ASN1_SEQUENCE_OF(SCVP_CERTREPLY, replyWantBacks, SCVP_REPLYWANTBACK),
  ASN1_SIMPLE(SCVP_CERTREPLY, valPolicy, SCVP_VALIDPOLICY),
  ASN1_IMP_OPT(SCVP_CERTREPLY, nextUpdate, ASN1_GENERALIZEDTIME, 1),
  ASN1_IMP_SEQUENCE_OF_OPT(SCVP_CERTREPLY, certReplyExtensions, X509_EXTENSION, 2)
} ASN1_SEQUENCE_END(SCVP_CERTREPLY)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_CERTREPLY)

ASN1_SEQUENCE(SCVP_HASHVALUE) = {
  ASN1_SIMPLE(SCVP_HASHVALUE, algorithm, X509_ALGOR),
  ASN1_SIMPLE(SCVP_HASHVALUE, value, ASN1_OCTET_STRING)
} ASN1_SEQUENCE_END(SCVP_HASHVALUE)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_HASHVALUE)

ASN1_CHOICE(SCVP_REQUEST_REFERENCE) = {
  ASN1_IMP(SCVP_REQUEST_REFERENCE, value.requestHash, SCVP_HASHVALUE, 1),
  ASN1_IMP(SCVP_REQUEST_REFERENCE, value.fullRequest, SCVP_REQUEST, 2)
} ASN1_CHOICE_END(SCVP_REQUEST_REFERENCE)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_REQUEST_REFERENCE)

ASN1_SEQUENCE(SCVP_RESPONSE_STATUS) = {
  ASN1_SIMPLE(SCVP_RESPONSE_STATUS, statusCode, ASN1_ENUMERATED),
  ASN1_IMP_OPT(SCVP_RESPONSE_STATUS, errorMessage, ASN1_UTF8STRING, 0)
} ASN1_SEQUENCE_END(SCVP_RESPONSE_STATUS)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_RESPONSE_STATUS)

ASN1_SEQUENCE(SCVP_RESPONSE) = {
  ASN1_SIMPLE(SCVP_RESPONSE, scvpVersion, ASN1_INTEGER),
  ASN1_SIMPLE(SCVP_RESPONSE, producedAt, ASN1_GENERALIZEDTIME),
  ASN1_SIMPLE(SCVP_RESPONSE, responseStatus, SCVP_RESPONSE_STATUS),
  ASN1_SIMPLE(SCVP_RESPONSE, requestRef, SCVP_REQUEST_REFERENCE),
  ASN1_IMP_OPT(SCVP_RESPONSE, requestor, ASN1_OCTET_STRING, 1),
  ASN1_IMP_OPT(SCVP_RESPONSE, responder, ASN1_OCTET_STRING, 2),
  ASN1_IMP_SEQUENCE_OF_OPT(SCVP_RESPONSE, replyObjects, SCVP_CERTREPLY, 3),
  ASN1_IMP_OPT(SCVP_RESPONSE, requestNonce, ASN1_OCTET_STRING, 4),
  ASN1_IMP_OPT(SCVP_RESPONSE, serverContextInfo, ASN1_OCTET_STRING, 5),
  ASN1_IMP_SEQUENCE_OF_OPT(SCVP_RESPONSE, respExtensions, X509_EXTENSION, 6)
} ASN1_SEQUENCE_END(SCVP_RESPONSE)

IMPLEMENT_ASN1_FUNCTIONS(SCVP_RESPONSE)
```

Annexe C

Code du client SCVP

Ci-après, le code du client SCVP qui se base sur mon implémentation du protocole SCVP dans OpenSSL (openssl/scvp.h).

```
#include <sys/param.h>
#include <err.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <curl/curl.h>
#include <curl/types.h>
#include <curl/easy.h>
#include <openssl/rand.h>
#include <openssl/scvp.h>

#define NONCE_SIZE 4

#define DEF_CHECKS_OID ID_STC_BUILD_STATUS_CHECKED_PKC_PATH
#define DEF_WANTBACK_OID ID_SWB_PKC_CERT_STATUS
#define DEF_VALPOLICY_OID ID_SVP_DEFAULT_VALPOLICY
#define AI_SHA_1 "1.3.14.3.2.26"

struct MemoryStruct {
char *memory;
size_t size;
} chunk;

SCVP_REQUEST *CVRequest;
SCVP_RESPONSE *CVResponse;
```

```
int scvp_cert_verify(X509 *cert, char *url);
void scvp_build_request(SCVP_REQUEST *req, X509 *cert);
int scvp_check_response(SCVP_RESPONSE *resp, SCVP_REQUEST *req, X509 *cert,
unsigned char *sbuf, int len);
size_t WriteMemory_cb(void *ptr, size_t size, size_t nmemb, void *data);
char ebuf[CURL_ERROR_SIZE];

int
scvp_cert_verify(X509 *cert, char *url)
{
CURL *curl;
unsigned char *sbuf, *p;
struct curl_slist *hlist = NULL;
int len, cr;

CVRequest = SCVP_REQUEST_new();
if (CVRequest == NULL)
errx(1, "can't allocate request");
scvp_build_request(CVRequest, cert);
len = i2d_SCVP_REQUEST(CVRequest, NULL);
if (len <= 0)
errx(1, "request encoding failed (get length)");
sbuf = (unsigned char *) malloc(len + 1);
bzero(sbuf, len + 1);
p = sbuf;
if (i2d_SCVP_REQUEST(CVRequest, &p) <= 0)
errx(1, "request encoding failed (real encoding)");

/* debug */
{
int fd = open("req", O_WRONLY|O_CREAT|O_TRUNC, 0644);

(void) write(fd, (char *) sbuf, len);
(void) close(fd);
}

curl = curl_easy_init();
if (curl == NULL)
errx(1, "curl init failed");
if (curl_easy_setopt(curl, CURLOPT_URL, url) != CURLE_OK)
errx(1, "curl setopt failed (URL)");
if (curl_easy_setopt(curl, CURLOPT_POSTFIELDS,
     (char *) sbuf) != CURLE_OK)
errx(1, "curl setopt failed (POST)");
if (curl_easy_setopt(curl, CURLOPT_POSTFIELDSIZE, len) != CURLE_OK)
errx(1, "curl setopt failed (SIZE)");
```

```
hlist = curl_slist_append(hlist,
  "Content-Type: application/scvp-request");
if (hlist == NULL)
errx(1, "curl header append failed");
if (curl_easy_setopt(curl, CURLOPT_HTTPHEADER, hlist) != CURLE_OK)
errx(1, "curl setopt failed (HEADER)");
if (curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,
     &WriteMemory_cb) != CURLE_OK)
errx(1, "curl setopt failed (WRITECB)");
if (curl_easy_setopt(curl, CURLOPT_FILE, (void *) &chunk) != CURLE_OK)
errx(1, "curl setopt failed (WRITE)");
if (curl_easy_setopt(curl, CURLOPT_ERRORBUFFER, ebuf) != CURLE_OK)
errx(1, "curl setopt failed (ERROR)");
if (curl_easy_perform(curl) != CURLE_OK)
errx(1, "curl perform failed: %s", ebuf);

/* should check the Content-Type: application/scvp-response */

p = (unsigned char *) chunk.memory;
CVResponse = d2i_SCVP_RESPONSE(NULL, &p, chunk.size);
if (CVResponse == NULL)
errx(1, "response decoding failed");
cr = scvp_check_response(CVResponse, CVRequest, cert, sbuf, len);

curl_global_cleanup();
curl_slist_free_all(hlist);
SCVP_REQUEST_free(CVRequest);
SCVP_RESPONSE_free(CVResponse);

return cr;
}

void
scvp_build_request(SCVP_REQUEST *req, X509 *cert)
{
SCVP_CERTREF *certRef;
ASN1_OBJECT *tmp_obj;
SCVP_PKCREF *pkc;
SCVP_VALIDPOLICY *valPol;
char *hostname;
unsigned char nonce[NONCE_SIZE];

if (req->scvpVersion == NULL)
errx(1, "build request: allocation failed (global)");
ASN1_INTEGER_set(req->scvpVersion, 1L);
```

```
/* Query */
if (req->query == NULL)
errx(1, "build request: allocation failed (global)");

/* Query -> queryCerts */
req->query->queriedCerts = sk_SCVP_CERTREF_new_null();
pkc = SCVP_PKCREF_new();
certRef = SCVP_CERTREF_new();
if ((req->query->queriedCerts == NULL) ||
    (pkc == NULL) || (certRef == NULL))
errx(1, "build request: allocation failed (queryCerts)");
pkc->type = V_SCVP_PKCREF_CERT;
pkc->value.cert = cert;
certRef->type = V_SCVP_CERTREF_PKC;
certRef->value.pkc = pkc;
if (sk_SCVP_CERTREF_push(req->query->queriedCerts, certRef) <= 0)
errx(1, "build request: allocation failed (push certRef)");

/* Query -> Checks */
req->query->checks = sk_ASN1_OBJECT_new_null();
if (req->query->checks == NULL)
errx(1, "build request: allocation failed (Checks)");
tmp_obj = OBJ_txt2obj(DEF_CHECKS_OID, 0);
if (tmp_obj == NULL)
errx(1, "build request: txt2obj failed");
if (sk_ASN1_OBJECT_push(req->query->checks, tmp_obj) <= 0)
errx(1, "build request: allocation failed (push check)");

        /* Query -> WantBacks */
  req->query->wantBack = sk_ASN1_OBJECT_new_null();
if (req->query->wantBack == NULL)
errx(1, "build request: allocation failed (WantBacks)");
tmp_obj = OBJ_txt2obj(DEF_WANTBACK_OID, 0);
if (tmp_obj == NULL)
errx(1, "build request: txt2obj failed");
if (sk_ASN1_OBJECT_push(req->query->wantBack, tmp_obj) <= 0)
errx(1, "build request: allocation failed (push wantback)");

/* Query -> ValPolicy */
valPol = req->query->valPolicy;
if (valPol == NULL)
errx(1, "build request: allocation failed (global)");
valPol->valPolicyId = OBJ_txt2obj(DEF_VALPOLICY_OID, 0);
if (valPol->valPolicyId == NULL)
errx(1, "build request: txt2obj failed");
```

```
/* Requestor */
req->requestor = ASN1_OCTET_STRING_new();
if (req->requestor == NULL)
errx(1, "build request: allocation failed (requestor)");
hostname = malloc(MAXHOSTNAMELEN + 1);
if (hostname == NULL)
errx(1, "build request: allocation failed (hostname)");
bzero(hostname, MAXHOSTNAMELEN + 1);
if (gethostname(hostname, MAXHOSTNAMELEN) < 0)
err(1, "build request: gethostname:");
ASN1_OCTET_STRING_set(req->requestor, hostname, strlen(hostname));

/* RequestNonce */
req->requestNonce = ASN1_OCTET_STRING_new();
if (req->requestNonce == NULL)
errx(1, "build request: allocation failed (requestNonce)");
if (RAND_pseudo_bytes(nonce, NONCE_SIZE) <= 0)
errx(1, "build request: can't get nonce");
ASN1_OCTET_STRING_set(req->requestNonce, nonce, NONCE_SIZE);

/* ReqExtensions */
/* req->reqExtensions = sk_X509_EXTENSION_new_null(); */
}

int
scvp_check_response(SCVP_RESPONSE *resp, SCVP_REQUEST *req, X509 *cert,
    unsigned char *sbuf, int slen)
{
SCVP_CERTREPLY *crep;
ASN1_OBJECT *tmp_obj;
ASN1_OCTET_STRING *tmp_ostr;
EVP_MD_CTX mdctx;
unsigned char md_value[EVP_MAX_MD_SIZE];
long status;
int md_len, i;

/* scvpVersion */
if ((resp->scvpVersion == NULL) ||
    ASN1_INTEGER_cmp(resp->scvpVersion, req->scvpVersion))
errx(1, "check response: bad version");

/* producedAt */

/* responseStatus */
if ((resp->responseStatus == NULL) ||
    (resp->responseStatus->statusCode == NULL))
```

```
errx(1, "check response: bad status");
status = ASN1_ENUMERATED_get(resp->responseStatus->statusCode);
if (status != SCVP_STATUS_CODE_OKAY) {
warnx("check response: status = %ld", status);
return (int) -status * 1000000;
}

/* requestRef */
if (resp->requestRef == NULL)
errx(1, "check response: bad requestRef");
switch (resp->requestRef->type) {
case V_SCVP_REQUEST_REFERENCE_REQUESTHASH:
/* should check the algorithm */
if (resp->requestRef->value.requestHash == NULL)
errx(1, "check response: no hash in requestRef");
tmp_ostr = resp->requestRef->value.requestHash->value;
if (tmp_ostr == NULL)
errx(1, "check response: void hash in requestRef");
EVP_MD_CTX_init(&mdctx);
EVP_DigestInit_ex(&mdctx, EVP_sha1(), NULL);
EVP_DigestUpdate(&mdctx, sbuf, slen);
EVP_DigestFinal_ex(&mdctx, md_value, &md_len);
EVP_MD_CTX_cleanup(&mdctx);
if ((md_len != tmp_ostr->length) ||
    bcmp(md_value, tmp_ostr->data, md_len)) {
warnx("check response: hash mismatch");
return -1;
}
break;
case V_SCVP_REQUEST_REFERENCE_FULLREQUEST:
if (resp->requestRef->value.fullRequest == NULL)
errx(1, "check response: void requestRef");
if (bcmp(resp->requestRef->value.fullRequest, sbuf, slen)) {
warnx("check response: requestRef mismatch");
return -1;
}
break;
default:
errx(1, "check response: unknown requestRef");
}

/* requestor */
if (resp->requestor == NULL)
errx(1, "check response: bad requestor");
if (ASN1_OCTET_STRING_cmp(resp->requestor, req->requestor))
errx(1, "check response: requestor mismatch");
```

```
/* responder */

/* replyObjects */

if ((resp->replyObjects == NULL) ||
    (sk_SCVP_CERTREPLY_num(resp->replyObjects) != 1))
errx(1, "check response: bad replyObjects");
crep = sk_SCVP_CERTREPLY_value(resp->replyObjects, 0);
if (crep == NULL)
errx(1, "check response: bad CertReply");

/* CertReply -> cert */
if ((crep->cert == NULL) ||
    (crep->cert->type != V_SCVP_CERTREF_PKC) ||
    (crep->cert->value.pkc == NULL) ||
    (crep->cert->value.pkc->type != V_SCVP_PKCREF_CERT) ||
    (crep->cert->value.pkc->value.cert == NULL) ||
    X509_cmp(crep->cert->value.pkc->value.cert, cert))
errx(1, "check reply: bad cert");

/* CertReply -> replyStatus */
if (crep->replyStatus == NULL)
errx(1, "check reply: bad status");
status = ASN1_ENUMERATED_get(crep->replyStatus);
if (status != SCVP_REPLY_STATUS_SUCCESS) {
warnx("check reply: status = %ld", status);
return (int) -status * 1000;
}

/* CertReply -> replyValTime */
if (crep->replyValTime == NULL)
errx(1, "check reply: bad valtime");
/* should do more checks */

/* CertReply -> replyChecks */
if (crep->replyChecks == NULL)
errx(1, "check reply: bad checks");
for (i = 0; i < sk_SCVP_REPLYCHECK_num(crep->replyChecks); i++) {
SCVP_REPLYCHECK *rcheck;

rcheck = sk_SCVP_REPLYCHECK_value(crep->replyChecks, i);
if ((rcheck == NULL) ||
    (rcheck->check == NULL) ||
    (rcheck->status == NULL)) {
warnx("check reply: void check %d", i);
```

```
continue;
}
/* should look at the check */
status = ASN1_INTEGER_get(rcheck->status);
if (status != 0L) {
warnx("check reply: bad status %ld in check %d",
      status, i);
return (int) -status * 10;
}
}

/* CertReply -> replyWantBacks */
if (crep->replyWantBacks == NULL)
errx(1, "check reply: bad wantbacks");
tmp_obj = OBJ_txt2obj(DEF_WANTBACK_OID, 0);
if (tmp_obj == NULL)
warnx("check reply: can't get default wantback OID");
for (i = 0; i < sk_SCVP_REPLYWANTBACK_num(crep->replyWantBacks); i++) {
SCVP_REPLYWANTBACK *rwback;
ASN1_BOOLEAN st;
unsigned char *p;

rwback = sk_SCVP_REPLYWANTBACK_value(crep->replyWantBacks, i);
if ((rwback == NULL) ||
    (rwback->wb == NULL) ||
    (rwback->value == NULL)) {
warnx("check reply: void wantback %d", i);
continue;
}
/* TODO: other wantbacks */
if ((tmp_obj == NULL) || OBJ_cmp(tmp_obj, rwback->wb))
continue;
p = rwback->value->data;
st = d2i_ASN1_BOOLEAN(NULL, &p, rwback->value->length);
if (st < 0)
errx(1, "check reply: can't get wantback value");
if (st == 0) {
warnx("check reply: wantback said cert is not valid");
return -1;
}
}
if (tmp_obj != NULL)
ASN1_OBJECT_free(tmp_obj);

/* CertReply -> valPolicy */
if ((crep->valPolicy == NULL) ||
```

```
    (crep->valPolicy->valPolicyId == NULL))
errx(1, "check reply: bad valPolicy");

/* requestNonce */
if (resp->requestNonce == NULL)
errx(1, "check response: bad requestNonce");
if (ASN1_OCTET_STRING_cmp(resp->requestNonce, req->requestNonce))
errx(1, "check response: requestNonce mismatch");

/* serverContextInfo */

/* respExtensions */

return 0;
}

size_t
WriteMemory_cb(void *ptr, size_t size, size_t nmemb, void *data)
{
register int realsize = size * nmemb;
struct MemoryStruct *mem = (struct MemoryStruct *)data;

mem->memory = (char *) realloc(mem->memory, mem->size + realsize + 1);
if (mem->memory) {
bcopy(ptr, &(mem->memory[mem->size]), realsize);
mem->size += realsize;
mem->memory[mem->size] = 0;
} else
err(1, "realloc failed:");
return realsize;
}

int
main(int argc, char *argv[])
{
BIO *in;
X509 *cert;
int cr;

/* ERR_load_crypto_strings(); */
/* OpenSSL_add_all_algorithms(); */

if (argc != 3)
errx(1, "usage: %s certfile URL", argv[0]);
in = BIO_new_file(argv[1], "r");
if (in == NULL)
```

```
err(1, "can't open %s", argv[1]);
cert = d2i_X509_bio(in, NULL);
if (cert == NULL)
errx(1, "can't decode cert");
cr = scvp_cert_verify(cert, argv[2]);
printf("cr = %d\n", cr);

exit(0);
}
```

Annexe D

Code du serveur SCVP

Ci-après, le code du serveur SCVP qui se base sur mon implémentation du protocole SCVP dans OpenSSL (openssl/scvp.h).

```
#include <sys/param.h>
#include <sys/socket.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <curl/curl.h>
#include <curl/types.h>
#include <curl/easy.h>
#include <openssl/err.h>
#include <openssl/rand.h>
#include <openssl/scvp.h>
#include <openssl/x509_vfy.h>

#ifdef no_debug
#define log(x,y)
#else
#define log(x,y) fprintf(stderr, x "\n", y)
#endif

#define DEF_CHECKS_OID ID_STC_BUILD_STATUS_CHECKED_PKC_PATH
#define DEF_WANTBACK_OID ID_SWB_PKC_CERT_STATUS
#define DEF_VALPOLICY_OID ID_SVP_DEFAULT_VALPOLICY
#define AI_SHA_1 "1.3.14.3.2.26"

typedef struct scvp_responder_ctx_st {
```

```
SCVP_REQUEST *CVRequest;
SCVP_RESPONSE *CVResponse;
SCVP_CERTREF *certRef;
X509 *cert;
long response_status;
long reply_status;
long check_status;
long wb_status;
unsigned char *rbuf, *sbuf;
int rlen, slen;
} SCVP_RESPONDER_CTX;

typedef struct global_context_st {
X509_STORE_CTX csc;
SCVP_RESPONDER_CTX ctx;
} SCVP_CTX;

char *scvp_srv(X509_STORE *store, int *lenp);
int cb(int ok, X509_STORE_CTX *csc);
void scvp_check_request(SCVP_RESPONDER_CTX *ctx);
int scvp_build_response(SCVP_RESPONDER_CTX *ctx);

X509_STORE *store;
X509_LOOKUP *lookup;
char *CAfile, *CApath;
char line[256];

int
main(int argc, char *argv[])
{
char *p, *e, *sbuf;
int len, cc, state, status = 0;

#define ERRX(code, msg, arg) { \
status = code; \
log(msg, arg); \
goto senderr; \
} while(0)

/* decode arguments */

CAfile = NULL;
CApath = NULL;

while (argc >= 2) {
if (*(argv[1] + strlen(argv[1]) - 1) == '/')
```

```
CApath = argv[1];
else
CAfile = argv[1];
argc -= 2;
}

ERR_load_crypto_strings();
OpenSSL_add_all_algorithms();

/* init store */

store = X509_STORE_new();
if (store == NULL)
ERRX(500, "can't allocate store", "");
X509_STORE_set_verify_cb_func(store, &cb);
lookup = X509_STORE_add_lookup(store, X509_LOOKUP_file());
if (lookup == NULL)
ERRX(500, "can't set file lookup", "");
if (CAfile) {
if (!X509_LOOKUP_load_file(lookup, CAfile, X509_FILETYPE_PEM))
ERRX(500, "can't load CAfile = %s", CAfile);
} else
X509_LOOKUP_load_file(lookup, NULL, X509_FILETYPE_DEFAULT);
lookup = X509_STORE_add_lookup(store, X509_LOOKUP_hash_dir());
if (lookup == NULL)
ERRX(500, "can't set directory lookup", "");
if (CApath) {
if (!X509_LOOKUP_add_dir(lookup, CApath, X509_FILETYPE_PEM))
ERRX(500, "can't load CApath = %s", CApath);
} else
X509_LOOKUP_add_dir(lookup, NULL, X509_FILETYPE_DEFAULT);
/* ADD LDAP method here ? */
/* DNSsec trust ? */
if (!X509_STORE_set_trust(store, X509_TRUST_COMPAT))
ERRX(500, "can't set compat trust", "");
ERR_clear_error();

/* parse headers */

for (state = 0, len = 0;;) {
bzero(line, 256);
cc = recv(0, line, 256, MSG_PEEK);
if (cc <= 0)
ERRX(400, "can't read: %s", strerror(errno));
e = line + cc;
if ((line[0] == '\r') && (line[1] == '\n')) {
```

```
/* end of headers */
if (len == 0)
ERRX(411, "no Content-Length", "");
if (state == 3) {
(void) read(0, line, 2);
break;
}
ERRX(412, "bad headers", "");
}
if (state == 0) {
/* init: get POST */
if (strncmp(line, "POST ", 5) == 0) {
state = 1;
goto next;
} else
ERRX(405, "can't get POST", "");
}
/* get Content-xxx: */
if (strncmp(line, "Content-", 8) != 0)
goto next;
/* Content-Length: <len> */
if ((len == 0) &&
    (strncmp(line, "Content-Length:", 15) == 0)) {
len = (int)strtol(line + 15, NULL, 10);
if (len <= 0)
ERRX(412, "bad length %d", len);
if (state == 2)
state = 3;
goto next;
}
/* Content-Type: <type> */
if (strncmp(line, "Content-Type:", 13) == 0) {
p = line + 13;
while (p <= e - 26) {
if ((*p == ' ') | (*p == '\t'))
p++;
else
break;
}
if (strncmp(p, "application/scvp-request", 24) == 0) {
state = 2;
goto next;
}
ERRX(415, "bad content type", "");
}
    next:
```

```
/* get end of line */
p = line + 1;
while (p < e) {
if ((*p != '\r') || (*(p + 1) != '\n'))
p++;
else
break;
}
p++;
if (p >= e)
ERRX(400, "line too long", "");
(void) read(0, line, p - line + 1);
}

#undef ERRX

/* do it */

sbuf = scvp_srv(store, &len);
if (sbuf == NULL) {
status = len;
    senderr:
printf("HTTP/1.1 %03d fatal error\r\n", status);
exit(1);
} else {
printf("HTTP/1.1 200 OK\r\n");
printf("Content-Length: %d\r\n", len);
printf("Content-Type: application/scvp-response\r\n");
printf("\r\n");
fflush(stdout);

for (cc = 0; len > 0; len -= cc) {
cc = write(1, (char *)sbuf, len);
if (cc <= 0) {
log("can't write: %s", strerror(errno));
exit(1);
}
}
exit(0);
}
}

char *
scvp_srv(X509_STORE *store, int *lenp)
{
SCVP_CTX *gcp;
```

```
X509_STORE_CTX *csc;
SCVP_RESPONDER_CTX *ctx;
unsigned char *p;
int cc;

#define ERRX(code, msg, arg) { \
log(msg, arg); \
*lenp = code; \
return NULL; \
}

gcp = (SCVP_CTX *) malloc(sizeof(SCVP_CTX));
if (gcp == NULL)
ERRX(500, "can't allocate contexts", strerror(errno));
bzero(gcp, sizeof(SCVP_CTX));
csc = &gcp->csc;
ctx = &gcp->ctx;
ctx->response_status = 0;
ctx->reply_status = -1;
ctx->check_status = -1;
ctx->wb_status = -1;
ctx->rlen = *lenp;

ctx->CVResponse = SCVP_RESPONSE_new();
if (ctx->CVResponse == NULL)
ERRX(500, "can't create response", "");

ctx->rbuf = (unsigned char *) malloc(ctx->rlen);
if (ctx->rbuf == NULL)
ERRX(500, "can't allocate receive buffer:", strerror(errno));
for (p = ctx->rbuf, cc = 0; p != ctx->rbuf + ctx->rlen; p += cc) {
cc = read(0, (char *)p, (p - ctx->rbuf) + ctx->rlen);
if (cc < 0)
ERRX(500, "can't read:", strerror(errno));
if (cc == 0)
ERRX(500, "short read: missing %d",
     (p - ctx->rbuf) + ctx->rlen);
}

p = ctx->rbuf;
ctx->CVRequest = d2i_SCVP_REQUEST(NULL, &p, ctx->rlen);
if (ctx->CVRequest == NULL)
ERRX(500, "request decoding failed", "");

scvp_check_request(ctx);
if ((ctx->response_status > 0) || (ctx->reply_status > 0))
```

```
goto sendit;

X509_STORE_CTX_init(csc, store, ctx->cert, NULL);
if (!X509_verify_cert(csc))
ctx->wb_status = 0;
else
ctx->wb_status = 1;

   sendit:
if (ctx->wb_status < 0)
ctx->wb_status = 0;
if (ctx->reply_status < 0)
ctx->reply_status = 0;
if (ctx->check_status < 0)
ctx->check_status = 0;
if (ctx->response_status < 0)
ctx->response_status = 0;

if (scvp_build_response(ctx) < 0)
ERRX(500, "can't build response", "");
ctx->slen = i2d_SCVP_RESPONSE(ctx->CVResponse, NULL);
if (ctx->slen <= 0)
ERRX(500, "response encoding failed (get length)", "");
ctx->sbuf = (unsigned char *) malloc(ctx->slen + 1);
bzero(ctx->sbuf, ctx->slen + 1);
p = ctx->sbuf;
if (i2d_SCVP_RESPONSE(ctx->CVResponse, &p) <= 0)
ERRX(500, "response encoding failed (real encoding)", "");

#undef ERRX
/* debug */
{
int fd = open("resp", O_WRONLY|O_CREAT|O_TRUNC, 0644);

(void) write(fd, (char *) ctx->sbuf, ctx->slen);
(void) close(fd);
}

*lenp = ctx->slen;
return (char *)ctx->sbuf;
}

int
cb(int ok, X509_STORE_CTX *csc)
{
SCVP_RESPONDER_CTX *ctx = &((SCVP_CTX *)csc)->ctx;
```

```
if (ok)
return ok;

switch (csc->error) {
case X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT:
case X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY:
case X509_V_ERR_PATH_LENGTH_EXCEEDED:
case X509_V_ERR_UNABLE_TO_DECODE_ISSUER_PUBLIC_KEY:
case X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE:
case X509_V_ERR_CERT_CHAIN_TOO_LONG:
case X509_V_ERR_SUBJECT_ISSUER_MISMATCH:
case X509_V_ERR_AKID_SKID_MISMATCH:
case X509_V_ERR_AKID_ISSUER_SERIAL_MISMATCH:
case X509_V_ERR_KEYUSAGE_NO_CERTSIGN:
ctx->reply_status = SCVP_REPLY_STATUS_CERTPATHCONSTRUCTFAIL;
ctx->check_status = 2;
ctx->wb_status = 0;
break;

case X509_V_ERR_UNABLE_TO_GET_CRL:
case X509_V_ERR_UNABLE_TO_GET_CRL_ISSUER:
case X509_V_ERR_UNABLE_TO_DECRYPT_CRL_SIGNATURE:
case X509_V_ERR_ERROR_IN_CRL_LAST_UPDATE_FIELD:
case X509_V_ERR_ERROR_IN_CRL_NEXT_UPDATE_FIELD:
case X509_V_ERR_CRL_NOT_YET_VALID:
case X509_V_ERR_CRL_HAS_EXPIRED:
case X509_V_ERR_CERT_NOT_YET_VALID:
case X509_V_ERR_CRL_SIGNATURE_FAILURE:
ctx->reply_status = SCVP_REPLY_STATUS_CERTPATHNOTVALIDNOW;
ctx->check_status = 3;
ctx->wb_status = 0;
break;

case X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT:
case X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN:
case X509_V_ERR_INVALID_CA:
case X509_V_ERR_CERT_SIGNATURE_FAILURE:
case X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD:
case X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD:
case X509_V_ERR_CERT_HAS_EXPIRED:
case X509_V_ERR_UNABLE_TO_DECRYPT_CERT_SIGNATURE:
ctx->reply_status = SCVP_REPLY_STATUS_CERTPATHNOTVALID;
ctx->check_status = 2;
ctx->wb_status = 0;
break;
```

```
case X509_V_ERR_CERT_REVOKED:
ctx->check_status = 1;
case X509_V_ERR_APPLICATION_VERIFICATION:
case X509_V_ERR_UNHANDLED_CRITICAL_EXTENSION:
case X509_V_ERR_INVALID_PURPOSE:
case X509_V_ERR_CERT_UNTRUSTED:
case X509_V_ERR_CERT_REJECTED:
ctx->wb_status = 0;
break;

default:
ctx->wb_status = 0;
case X509_V_ERR_OUT_OF_MEM:
ctx->response_status = SCVP_STATUS_CODE_INTERNALERROR;
break;
}
log("cb called for error %d", csc->error);
ERR_clear_error();

return ok;
}

void
scvp_check_request(SCVP_RESPONDER_CTX *ctx)
{
SCVP_REQUEST *req = ctx->CVRequest;
SCVP_RESPONSE *resp = ctx->CVResponse;
SCVP_QUERY *query;
ASN1_OBJECT *tmp_obj = NULL;

#define getRerr(x) { \
ctx->response_status = (x); \
        return; \
} while(0)

#define getQerr(x) { \
ctx->reply_status = (x); \
if (tmp_obj != NULL) ASN1_OBJECT_free(tmp_obj); \
        return; \
} while(0)

/* scvpVersion */
if (req->scvpVersion == NULL)
getRerr(SCVP_STATUS_CODE_UNABLETODECODE);
if (ASN1_INTEGER_get(req->scvpVersion) != 1L)
```

```
getRerr(SCVP_STATUS_CODE_UNSUPPORTEDVERSION);

/* requestor */
if (req->requestor != NULL)
resp->requestor = ASN1_OCTET_STRING_dup(req->requestor);

/* requestNonce */
if (req->requestNonce != NULL)
resp->requestNonce = ASN1_OCTET_STRING_dup(req->requestNonce);

/* reqExtensions */
if ((req->reqExtensions != NULL) &&
    (sk_X509_EXTENSION_num(req->reqExtensions) > 0) &&
    (sk_X509_EXTENSION_value(req->reqExtensions, 0) != NULL))
getRerr(SCVP_STATUS_CODE_SKIPUNRECOGNIZEDITEMS);

/* query */
query = req->query;
if (query == NULL)
getRerr(SCVP_STATUS_CODE_BADSTRUCTURE);

/* query -> queriedCerts */
if ((query->queriedCerts == NULL) ||
    (sk_SCVP_CERTREF_num(query->queriedCerts) != 1) ||
    (sk_SCVP_CERTREF_value(query->queriedCerts, 0) == NULL))
getRerr(SCVP_STATUS_CODE_BADSTRUCTURE);
ctx->certRef = sk_SCVP_CERTREF_value(query->queriedCerts, 0);
if (ctx->certRef->type != V_SCVP_CERTREF_PKC)
getRerr(SCVP_STATUS_CODE_ABORTUNRECOGNIZEDITEMS);
if (ctx->certRef->value.pkc->type != V_SCVP_PKCREF_CERT)
getRerr(SCVP_STATUS_CODE_ABORTUNRECOGNIZEDITEMS);
ctx->cert = ctx->certRef->value.pkc->value.cert;
if (ctx->cert == NULL)
getQerr(SCVP_REPLY_STATUS_MALFORMEDPKC);

/* query -> checks */
if ((query->checks == NULL) ||
    (sk_ASN1_OBJECT_num(query->checks) != 1) ||
    (sk_ASN1_OBJECT_value(query->checks, 0) == NULL))
getRerr(SCVP_STATUS_CODE_UNSUPPERTEDCHECKS);
tmp_obj = OBJ_txt2obj(DEF_CHECKS_OID, 0);
if (tmp_obj == NULL)
getRerr(SCVP_STATUS_CODE_INTERNALERROR);
if (OBJ_cmp(tmp_obj, sk_ASN1_OBJECT_value(query->checks, 0)))
getQerr(SCVP_REPLY_STATUS_UNRECOGNIZEDCHECK);
```

```
/* query -> wantBack */
if ((query->wantBack == NULL) ||
    (sk_ASN1_OBJECT_num(query->wantBack) != 1) ||
    (sk_ASN1_OBJECT_value(query->wantBack, 0) == NULL))
getRerr(SCVP_STATUS_CODE_UNSUPORTEDWANTBACKS);
tmp_obj = OBJ_txt2obj(DEF_WANTBACK_OID, 0);
if (tmp_obj == NULL)
getRerr(SCVP_STATUS_CODE_INTERNALERROR);
if (OBJ_cmp(tmp_obj, sk_ASN1_OBJECT_value(query->wantBack, 0)))
getQerr(SCVP_REPLY_STATUS_UNRECOGNIZEDWANTBACK);

/* query -> validityTime */
if (query->validityTime != NULL)
getQerr(SCVP_REPLY_STATUS_UNAVAILABLEVALIDITYTIME);

/* query -> trustAnchors */
if ((query->trustAnchors != NULL) &&
    (sk_SCVP_TRUSTANCHOR_num(query->trustAnchors) > 0) &&
    (sk_SCVP_TRUSTANCHOR_value(query->trustAnchors, 0) != NULL))
getQerr(SCVP_REPLY_STATUS_UNRECOGNIZEDCERTPOLICY);

/* query -> queryExtensions */
if ((query->queryExtensions != NULL) &&
    (sk_X509_EXTENSION_num(query->queryExtensions) > 0) &&
    (sk_X509_EXTENSION_value(query->queryExtensions, 0) != NULL))
getQerr(SCVP_REPLY_STATUS_UNRECOGNIZEDEXTENSION);

/* query -> valPolicy */
if ((query->valPolicy == NULL) ||
    (query->valPolicy->valPolicyId == NULL))
getQerr(SCVP_REPLY_STATUS_SUCCESS);
tmp_obj = OBJ_txt2obj(DEF_VALPOLICY_OID, 0);
if (tmp_obj == NULL)
getRerr(SCVP_STATUS_CODE_INTERNALERROR);
if (OBJ_cmp(tmp_obj, query->valPolicy->valPolicyId))
getQerr(SCVP_REPLY_STATUS_UNRECOGNIZEDCERTPOLICY);

#undef getRerr
#undef getQerr

ctx->reply_status = SCVP_REPLY_STATUS_SUCCESS;
if (tmp_obj != NULL)
ASN1_OBJECT_free(tmp_obj);
        return;
}
```

```
int
scvp_build_response(SCVP_RESPONDER_CTX *ctx)
{
SCVP_RESPONSE *resp = ctx->CVResponse;
SCVP_REQUEST *req = ctx->CVRequest, *creq;
SCVP_HASHVALUE *hv;
SCVP_CERTREPLY *reply;
SCVP_REPLYCHECK *check;
SCVP_REPLYWANTBACK *wantback;
EVP_MD_CTX mdctx;
ASN1_BOOLEAN st;
unsigned char md_value[EVP_MAX_MD_SIZE], *p;
char *hostname;
int md_len;

#define ERRX(fatal, msg, arg) { \
ctx->response_status = SCVP_STATUS_CODE_INTERNALERROR; \
log(msg, arg); \
if (fatal) return -1; \
goto finish; \
}

/* scvpVersion */
if (resp->scvpVersion == NULL)
ERRX(1, "build response: allocation failed (global)", "");
ASN1_INTEGER_set(resp->scvpVersion, 1L);

/* produceAt */
if (resp->producedAt == NULL)
ERRX(1, "build response: allocation failed (global)", "");
ASN1_GENERALIZEDTIME_set(resp->producedAt, time(NULL));

/* requestRef */
if (resp->requestRef == NULL)
ERRX(0, "build response: allocation failed (global)", "");
creq = SCVP_REQUEST_dup(req);
resp->requestRef->type = V_SCVP_REQUEST_REFERENCE_REQUESTHASH;
hv = SCVP_HASHVALUE_new();
if (hv == NULL)
ERRX(0, "build response: allocation failed (requestRef)", "");
if (hv->algorithm)
ASN1_OBJECT_free(hv->algorithm->algorithm);
hv->algorithm->algorithm = OBJ_txt2obj(AI_SHA_1, 0);
if (hv->algorithm->algorithm == NULL)
ERRX(0, "build response: allocation failed (hash algo)", "");
EVP_MD_CTX_init(&mdctx);
```

```
EVP_DigestInit_ex(&mdctx, EVP_sha1(), NULL);
EVP_DigestUpdate(&mdctx, ctx->rbuf, ctx->rlen);
EVP_DigestFinal_ex(&mdctx, md_value, &md_len);
EVP_MD_CTX_cleanup(&mdctx);
ASN1_OCTET_STRING_set(hv->value, md_value, md_len);
if (hv->value == NULL)
ERRX(0, "build response: allocation failed (hash value)", "");
resp->requestRef->value.requestHash = hv;

/* responder */
resp->responder = ASN1_OCTET_STRING_new();
if (resp->responder == NULL)
ERRX(0, "build response: allocation failed (responder)", "");
hostname = malloc(MAXHOSTNAMELEN + 1);
if (hostname == NULL)
ERRX(0, "build response: allocation failed (hostname)", "");
if (gethostname(hostname, MAXHOSTNAMELEN) < 0)
ERRX(0, "build response: gethostname:", strerror(errno));
ASN1_OCTET_STRING_set(resp->responder, hostname, strlen(hostname));

/* replyObjects */
if (ctx->response_status > 0)
goto finish;
resp->replyObjects = sk_SCVP_CERTREPLY_new_null();
reply = SCVP_CERTREPLY_new();
if ((resp->replyObjects == NULL) ||
    (reply == NULL) ||
    (reply->cert == NULL) ||
    (reply->replyStatus == NULL) ||
    (reply->replyValTime == NULL) ||
    (reply->valPolicy == NULL) ||
    (reply->valPolicy->valPolicyId == NULL) ||
    (sk_SCVP_CERTREPLY_push(resp->replyObjects, reply) <= 0))
ERRX(0, "build response: allocation failed (reply)", "");

/* reply -> cert */
/* TODO: deal with pointers too */
SCVP_CERTREF_free(reply->cert);
reply->cert = SCVP_CERTREF_dup(ctx->certRef);

/* reply -> replyStatus */
ASN1_ENUMERATED_set(reply->replyStatus, ctx->reply_status);

/* reply -> replyValTime */
ASN1_GENERALIZEDTIME_set(reply->replyValTime, time(NULL));
```

```
/* reply -> replyChecks */
reply->replyChecks = sk_SCVP_REPLYCHECK_new_null();
check = SCVP_REPLYCHECK_new();
if ((reply->replyChecks == NULL) ||
    (check == NULL) ||
    (check->check == NULL) ||
    (check->status == NULL) ||
    (sk_SCVP_REPLYCHECK_push(reply->replyChecks, check) <= 0))
ERRX(0, "build response: allocation failed (check)", "");
check->check = OBJ_txt2obj(DEF_CHECKS_OID, 0);
ASN1_INTEGER_set(check->status, ctx->check_status);

/* reply -> replyWantBacks */
reply->replyWantBacks = sk_SCVP_REPLYWANTBACK_new_null();
wantback = SCVP_REPLYWANTBACK_new();
if ((reply->replyWantBacks == NULL) ||
    (wantback == NULL) ||
    (wantback->wb == NULL) ||
    (sk_SCVP_REPLYWANTBACK_push(reply->replyWantBacks, wantback) <= 0))
ERRX(0, "build response: allocation failed (wantback)", "");
wantback->wb = OBJ_txt2obj(DEF_WANTBACK_OID, 0);
st = ctx->wb_status ? 0xff : 0;
p = md_value;
md_len = i2d_ASN1_BOOLEAN(st, &p);
ASN1_OCTET_STRING_set(wantback->value, md_value, md_len);

/* reply -> valPolicy */
reply->valPolicy->valPolicyId = OBJ_txt2obj(DEF_VALPOLICY_OID, 0);
if (reply->valPolicy->valPolicyId == NULL)
ERRX(0, "build response: txt2obj failed", "");

/* responseStatus (must be last) */
   finish:
if ((resp->responseStatus == NULL) ||
    (resp->responseStatus->statusCode == NULL))
ERRX(1, "build response: allocation failed (global)", "");
ASN1_ENUMERATED_set(resp->responseStatus->statusCode,
    ctx->response_status);
#undef ERRX

return 0;
}
```

Annexe E

Exemple de requête/réponse ASN.1

Ci dessous, un extrait d'une requête SCVP après encodage en ASN.1 :

```
  0 1401: SEQUENCE {
  4    1:   INTEGER 1
  7 1354:   SEQUENCE {
 11 1314:     SEQUENCE {
 15 1310:       [1] {
 19 1030:         SEQUENCE {
 23    3:           [0] {
 25    1:             INTEGER 2
       :             }
 28    1:           INTEGER 1
 31   13:           SEQUENCE {
 33    9:             OBJECT IDENTIFIER
       :               sha1withRSAEncryption (1 2 840 113549 1 1 5)
 44    0:             NULL
       :             }

              ... Certificat ...

       :         }
       :       }
1329   10:     SEQUENCE {
1331    8:       OBJECT IDENTIFIER '1 3 6 1 5 5 7 17 3'
       :       }
1341   10:     SEQUENCE {
1343    8:       OBJECT IDENTIFIER '1 3 6 1 5 5 7 18 3'
       :       }
1353   10:     [1] {
```

```
1355    8:       OBJECT IDENTIFIER '1 3 6 1 5 5 7 19 1'
         :       }
         :     }
1365   32:   [0] 'ps2.ipv6.rennes.enst-bretagne.fr'
1399    4:   [1] D6 E8 AF 71
         :   }
```

Et la réponse du serveur SCVP à cette requête :

```
   0 1522: SEQUENCE {
   4    1:   INTEGER 1
   7   15:   GeneralizedTime 27/07/2003 18:17:26 GMT
  24    3:   SEQUENCE {
  26    1:     ENUMERATED 0
         :     }
  29   31:   [1] {
  31    7:     SEQUENCE {
  33    5:       OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
         :       }
  40   20:     OCTET STRING
         :       34 F8 38 64 6E 13 22 77 E5 7E E8 FA 92 8E 90 77
         :       FB E8 6A E3
         :     }
  62   32:   [1] 'ps2.ipv6.rennes.enst-bretagne.fr'
  96   32:   [2] 'ps2.ipv6.rennes.enst-bretagne.fr'
 130 1386:   [3] {
 134 1382:     SEQUENCE {
 138 1310:       [1] {
 142 1030:         SEQUENCE {
 146    3:           [0] {
 148    1:             INTEGER 2
         :             }
 151    1:           INTEGER 1
 154   13:           SEQUENCE {
 156    9:             OBJECT IDENTIFIER
         :               sha1withRSAEncryption (1 2 840 113549 1 1 5)
 167    0:             NULL
         :             }

               ... Certificat ...

1189    0:           NULL
         :           }

1191  257:         BIT STRING
         :           55 F4 95 82 7D 68 7B 6D 09 8E 57 11 98 DD 62 41
         :           85 B7 40 3A 6C 74 4D 68 AA 6A F2 AF D0 ED 9E BC
```

```
        :           CE 88 46 C9 47 FF 03 11 06 DA 1A 02 CE 1E F3 2E
        :           6E 18 66 AE 53 D3 9C 80 26 86 06 95 EF 29 AE F9
        :           82 4A A2 44 01 50 0F B6 1A 20 82 27 BA 3E D6 4F
        :           80 17 EB 1E 88 FD 85 4A B2 08 BC C4 A7 1D 54 2F
        :           E7 C2 FF FC 89 94 1C A2 D1 15 AC BF 7E 7E D9 D2
        :           0C DF 3D 9E BD B3 59 B7 1B 10 C4 3A 44 0E 70 C1
        :                   [ Another 128 bytes skipped ]
        :         }
1452    1:       ENUMERATED 0
1455   15:       GeneralizedTime 27/07/2003 18:17:26 GMT
1472   15:       SEQUENCE {
1474   13:         SEQUENCE {
1476    8:           OBJECT IDENTIFIER '1 3 6 1 5 5 7 17 3'
1486    1:           INTEGER 0
        :           }
        :         }
1489   17:       SEQUENCE {
1491   15:         SEQUENCE {
1493    8:           OBJECT IDENTIFIER '1 3 6 1 5 5 7 18 3'
1503    3:           OCTET STRING, encapsulates {
1505    1:             BOOLEAN TRUE
        :             }
        :           }
        :         }
1508   10:       SEQUENCE {
1510    8:         OBJECT IDENTIFIER '1 3 6 1 5 5 7 19 1'
        :         }
        :       }
        :     }
1520    4:   [4] D6 E8 AF 71
        :   }
```

Bibliographie

[1] A. Malpani, R. Housley, T. Freeman, *Draft IEFT of Simple Certificate Validation Protocol (SCVP)*, draft-ietf-pkix-scvp-12.txt, Internet draft, 2003.

[2] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, *RFC 2560 - Online Certificate Status Protocol (OCSP)*, IETF, 1999.

[3] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, *RFC 3280 - Internet X.509 Public Key Infrastructure*, Certificate and Cartificate Revocation List (CRL) Profile, R. Housley, W.Polk, W.Ford, D. Solo, Avril 2002.

[4] RSA Laboratories Technical Note, *PKCS #7 : Cryptographic Message Syntax Standard*, v1.5, RSA Laboratories, 1993.

[5] RSA Laboratories Technical Note, *PKCS #10 : Certification Request Syntax Standard*, v1.7, RSA Laboratories, 2000.

[6] RSA Laboratories Technical Note, *PKCS #12 : Personal Information Exchange Syntax*, v1.0, RSA Laboratories, 1999.

[7] ITU-T, *X.509 : Internet Public Key Infrastructure Certificate*, International telecommunication Union, 2002.

[8] ITU-T, *X.680 : Abstract Syntax Notation One (ASN.1)*, International telecommunication Union, 2002.

[9] ITU-T, *X.690 : BER/CER/DER Encoding for ASN.1*, International telecommunication Union, 2002.

[10] Burton S. Kaliski Jr., *A Layman's Guide to a Subset of ASN.1, BER and DER*, RSA Laboratories, 1993.

[11] N. Notari, J.-P. Pick, M. Souissi, *Projet fédérateur G6-DNSSEC*, AFNIC, 2002.

[12] O. Courtay, *IPsec : Présentation et Configuration*, IDsA, 2003.

[13] L. Bellefin, *Livre Blanc - Les PKI : Vers une infrastructure Globale de Sécurité*, SoluCom, 2001.

[14] idxpki, *La PKI en quelques mots*, IDEALX SAS, 2003.

[15] C. Starkjohann, *SSL Proxy - sslproxy.c*, GNU GPL, 1998.

[16] B. Bayart, *Joli Manuel pour LaTeX 2ε*, ESIEE, 1995.

[17] V. Seguin, *Aide-mémoire LaTeX*, ECP, 2000.