INTERNET DRAFT
draft-nourse-scep-04.txt
expires 23 August 2001

Xiaoyi Liu
Cheryl Madson
David McGrew
Andrew Nourse
Cisco Systems

Category: Informational

23 February 2001

Cisco Systems' Simple Certificate Enrollment Protocol(SCEP):

Status of this Memo

This document is an Internet-Draft and is NOT offered in accordance with Section 10 of RFC2026, and the author does not provide the IETF with any rights other than to publish as an Internet-Draft

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

This document specifies the Cisco Simple Certificate Enrollment Protocol, a PKI communication protocol which leverages existing technology by using PKCS#7 and PKCS#10. SCEP is the evolution of the enrollment protocol developed by Verisign, Inc. for Cisco Systems, Inc. It now enjoys wide support in both client and CA implementations.

Table of Contents

1. Introduction 2
2. The Goal of SCEP 3
2.1 SCEP Entity types 3
2.2 SCEP Operations Overview 7

2.3 PKI Operation Transactional Behavior 10
2.4 Security 12
3. Transport Protocol 13
4. Secure Transportation: PKCS #7 14
4.1 SCEP Message Format 14
4.2 Signed Transaction Attributes 15
5. SCEP Transaction Specification 16
6. Security Considerations 33
7. Intellectual Propoerty 33
8. References 33
Appendix A. Cisco End Entity Subject Name Definition 34
Appendix B. IPSEC Client Enrollment Certificate Request 35
Appendix C. Private OID Definitions 36
Appendix D. Obtaining CRL by LDAP Query 36
Appendix E. SCEP State Transitions 37
Appendix F. Author Contact Information. 40
Appendix G. Copyright Section 40

Section 1. Introduction

Public key technology is becoming more widely deployed and is becoming the basis for standards based security, such as the Internet Engineering Task Force's IPSEC and IKE protocols. With the use of public key certificates in network security protocols comes the need for a certificate management protocol that Public Key Infrastructure (PKI) clients and Certificate Authority servers can use to support certificate life cycle operations such as certificate enrollment and revocation, and certificate and CRL access.

In the following, Section 2 gives an overview of the PKI operations, and Section 2.4 describes the security goals of the protocol and the mechanisms used to achieve them. The transport protocol and the security protocol PKCS#7 are described at Section 3 and Section 4, respectively. The last section, Section 5, specifies each PKI operation in terms of the message formats and the data structures of each operation.

The appendices provide detailed specifications and examples. End entity subject names are specified in Appendix A, attribute OIDs are specified in Appendix C , and the SCEP state transitions are described in Appendix **E. An example of a certificate enrollment request is provided in** Appendix B, and an example LDAP query URL encoding is provided in Appendix D.

The authors would like to thank Peter William of ValiCert, Inc. (formerly of Verisign, Inc) and Alex Deacon of Verisign, Inc. and Christopher Welles of IRE, Inc. for their contributions to this protocol and to this document.

2.0 The Goal of SCEP

The goal of SCEP is to support the secure issuance of certificates to network devices in a scalable manner, using existing technology whenever possible. The protocol supports the following operations:

CA and RA public key distribution
Certificate enrollment
Certificate revocation
Certificate query
CRL query

Certificate and CRL access can be achieved by using the LDAP protocol (as specified in Appendix D), or by using the query messages defined in SCEP. The use of HTTP certificate and CRL access, and the support of CDP as specified in RFC2459, will be specified in a future version of this document. In Section 2.1, we first define PKI entity types as well as the properties of each entity type. In Section 2.2, the PKI operations are described at functional level. Section 2.3 describes the transaction behavior of each PKI operations. The complete PKI messages are covered in Section 5.

2.1 SCEP Entity types

The entity types defined in SCEP are the end entity type (i.e., IPSEC clients), the Certificate Authority (CA) entity type, and the Registration Authority entity type (RA). An end entity is sometimes called a "SCEP client" in the following.

2.1.1 End Entities

An end entity is an entity whose name is defined in a certificate subject name field and optionally, in SubjectAltName, a X.509 certificate V3 extension. As an end entity, a SCEP client is identified by a subject name consisting of the following naming attributes:

Fully qualified domain name, for example, router.cisco.com
IP address, or
Serial number.

In the paragraph above , the fully qualified domain name is required for each SCEP client, the IP address and the serial number are optional name attributes. In the certificate enrollment request, the PKCS#10 subject field contains the required and optional name attributes. Based on the PKCS#10 subject name information, the certificate issued to the SCEP client must have the same name attributes set both in the subjectName field and in the SubjectAltName extension.

It is important to note that a client named as Alice.cisco.com is different than a client named as Alice.cisco.com plus the IP address name attribute 171.69.1.129. From CA point of view, the Distinguished

names assigned in these two cases are distinct names.

Entity names which are specified as in the IPSEC profile (i.e., FQDN, IP address and User FQDN) must be presented in certificate's SubjectAltName extension. Multiple IPSEC entity names, (if any) are encoded as multiple values of a single SubjectAltName extension. The CA has the authority to assign a distinguished name to an end entity. The assigned DN should contain the SCEP client names as the relative DN.

The attribute identifiers and an example of SCEP client subject name are specified in Appendix A. Appendix B has an example from Cisco VPN Client enrollment request.

2.1.1.1 Local Key/Certificate/CRL Storage and Certificate-name uniqueness

An end entity is required to generate asymmetric key pairs and to provide storage to store its private keys. If the end entity does not have enough permanent memory to save its certificate, the end entity should be able to query its own certificate from the CA, once the certificate has been issued. The public key pairs can be generated with a specific key usage. The key usage are conveyed to the CA through the certificate enrollment request. All current SCEP client implementations expect that there will be only one pair of keys for a given subject name and key usage combination and CA, at any time. This property is called the certificate-name uniqueness property, and it implies that a CA that implements SCEP will enforce the unique mapping between a SCEP client subject name and its key pairs with a given key usage. At any time, if the subject name is changed, or if the key is updated, the existing certificate would have to be revoked before a new one could be issued.

It is desirable that the CA enforce certificate-name uniqueness, but it is not mandatory. However a CA that does not enforce uniqueness must provide some other mechanism to prevent the re-transmission of an enrollment request by a SCEP client from creating a second certificate or certificate request, nor can the second request merely be rejected. If a client times out from polling for a pending request it can resynchronize by reissuing the original request with the original subject name and transaction ID. This must return the status of the original transaction, including the certificate if it was granted. It must not create a new transaction unless the original cert has been revoked, or the transaction arrives more than halfway through the validity time of the original certificate.

An enrollment request that occurs more than halfway through the validity time of an existing certificate for the same subject name and key usage MAY be interpreted as a renewal request and accepted regardless of the duplication of subject name. Certificate renewal can be done this way.

2.1.1.2 End entity authentication

As with every protocol that uses public-key cryptography, the association between the public keys used in the protocol and the identities with which they are associated must be authenticated in a

cryptographically secure manner. This requirement is needed to prevent a "man in the middle" attack, in which an adversary that can manipulate the data as it travels between the protocol participants can subvert the security of the protocol. To satisfy this requirement, SCEP provides two authentication methods: manual authentication, and authentication based on pre-shared secret. In the manual mode, the end entity submitting the request is required to wait until its identity can be verified by the CA operator using any reliable out-of-band method. To prevent a "man-in-the-middle" attack, an MD5 `fingerprint' generated on the PKCS#10 (before PKCS #7 enveloping and signing) must be compared out-of-band between the server and the end entity. SCEP clients and CAs (or RAs, if appropriate) must display this fingerprint to a user to enable this verification, if manual mode is used. Failing to provide this information leaves the protocol vulnerable to attack by sophisticated adversaries. When utilizing a pre-shared secret scheme, the server should distribute a shared secret to the end entity which can uniquely associate the enrollment request with the given end entity. The distribution of the secret must be private: only the end entity should know this secret. The actual binding mechanism between the end entity and the secret is subject to the server policy and implementation. When creating enrollment request, the end entity is asked to provide a challenge password. When using the pre-shared secret scheme, the end entity must type in the re-distributed secret as the password. In the manual authentication case, the challenge password is also required since the server may challenge an end entity with the password before any certificate can be revoked. Later on, this challenge password will be included as a PKCS#10 attribute, and is sent to the server as encrypted data. The PKCS#7 envelope protects the privacy of the challenge password with DES encryption.

2.1.1.3 Self-Signed Certificates

In this protocol, the communication between the end entity and the certificate authority is secured by using PKCS#7 as the messaging protocol. PKCS#7, however, is a protocol which assumes the communicating entities already possess the peer's certificates and requires both parties use the issuer names and issuer assigned certificate serial numbers to identify the certificate in order to verify the signature and decrypt the message. When using PKCS#7 as a secure protocol for SCEP transactions this assumption may not be valid. To solve this problem, an end entity generates a self-signed certificate for its own public key. In this self-signed certificate, the issuer name is the end entity subject name (the same name later used in the PKCS#10). During the certificate enrollment, the end entity will first post itself as the signing authority by attaching the self-signed certificate to the signed certificate request. When the Certificate Authority makes the envelope on the issued certificate using the public key included in the self-signed certificate, it should use the same issuer name and serial

number as conveyed in the self-signed certificate to inform the end entity on which private key should be used to open the envelope.

Note that when a client enrolls for separate encryption and signature

certificates, it may use the signature certificate to sign both requests, and then expect its signature key to be used to encrypt both responses. In any case, the recipientinfo on the envelope should reflect the key used to encrypt the request.

2.1.2 Certificate Authority

A Certificate Authority(CA) is an entity whose name is defined in the certificate issuer name field. Before any PKI operations can begin, the CA generates its own public key pair and creates a self-signed CA certificate. Associated with the CA certificate is a fingerprint which will be used by the end entity to authenticate the received CA certificate. The fingerprint is created by calculating a MD5 hash on the whole CA certificate. Before any end entity can start its enrollment, this root certificate has to be configured at the entity side securely. For IPSEC clients, the client certificates must have SubjectAltName extension. To utilize LDAP as a CRL query protocol, the certificates must have CRL Distribution Point. Key usage is optional. Without key usage, the public key is assumed as a general purpose public key and it can be used for all the purposes.

A Certificate Authority may enforce certain name policy. When using **X.500 directory name as the subject name, all the name attributes** specified in the PKCS#10 request should be included as Relative DN. All the name attributes as defined in RFC2459 should be specified in the SubjectAltName. An example is provided in Appendix A.

If there is no LDAP query protocol support, the Certificate Authority should answer certificate and CRL queries, and to this end it should be online all the time.

The updating of the CA's public key is not addressed within the SCEP protocol. An SCEP client can remove its copy of a CA's public key and re-enroll under the CA's new public key.

2.1.3 Registration Authorities

In the environment where a RA is present, an end entity performs enrollment through the RA. In order to setup a secure channel with RA using PKCS#7, the RA certificate(s) have to be obtained by the client in addition to the CA certificate(s).

In the following, the CA and RA are specified as one entity in the context of PKI operation definitions.

2.1.4 Trusted Root Store

To support interoperability between IPSEC peers whose certificates are issued by different CA, SCEP allows the users to configure multiple trusted roots. A root is a trusted root when its certificate has been

configured as such in the client. An SCEP client that supports multiple roots must associate with each root the information needed to query a CRL from each root.

Once a trusted root is configured in the client, the client can verify the signatures of the certificates issued by the given root.

2.2 SCEP Operations Overview

In this section, we give a high level overview of the PKI operations as defined in SCEP.

2.2.1 End Entity Initialization

The end entity initialization includes the key pair generation and the configuring of the required information to communicate with the certificate authority.

2.2.1.1 Key Pair Generation

Before an end entity can start PKI transaction, it first generates asymmetric key pairs, using the selected algorithm (the RSA algorithm is required in SCEP, and is the only algorithm in current implementations).

An end entity can create one or more asymmetric key pairs, for different key usage. The key pairs can be created for encryption only, signing only, or for all purposes. For the same key usage, there can be only one key pair at any time.

The key pairs are saved by the client in NVRAM or other non-volatile media. The identification of a key pair is based on the FQDN assigned to the client and the selected key usage. Every time a new key pair is generated to replace the old key pair, the existing certificates have to be revoked from the CA and a new enrollment has to be completed.

2.2.1.2 Required Information

An end entity is required to have the following information configured before starting any PKI operations:

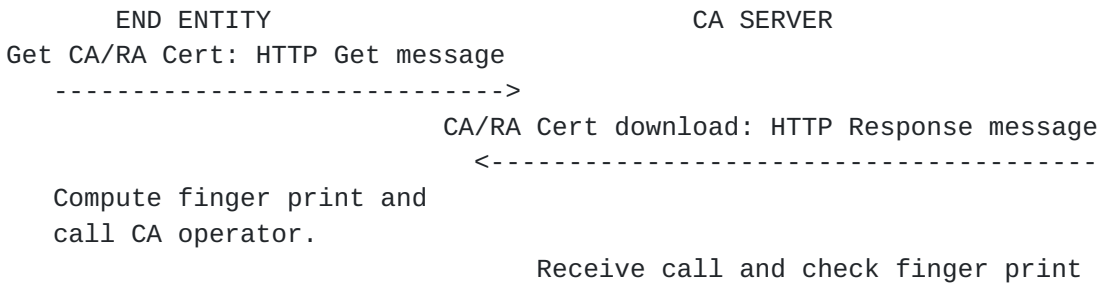
1. the certificate authority IP address or fully qualified domain name,
2. the certificate authority HTTP CGI script path, and
   the HTTP proxy information in case there is no direct Internet connection to the server,
3. the CRL query URL, if the CRL is to be obtained by from a directory server by means of LDAP.

2.2.2 CA/RA Certificate Distribution

Before any PKI operation can be started, the end entity needs to get the CA/RA certificates. At this time, since no public key has been

exchanged between the end entity and the CA/RA, the message to get the CA/RA certificate can not be secured using PKCS#7 protocol. Instead, the CA/RA certificate distribution is implemented as a clear HTTP Get operation. After the end entity gets the CA certificate, it has to authenticate the CA certificate by comparing the finger print with the CA/RA operator. Since the RA certificates are signed by the CA, there is no need to authenticate the RA certificates.

This operation is defined as a transaction consisting of one HTTP Get message and one HTTP Response message:



If an RA is in use, a degenerated PKCS#7 with a certificate chain consisting of both RA and CA certificates is sent back to the end entity. Otherwise the CA certificate is directly sent back as the HTTP response payload.

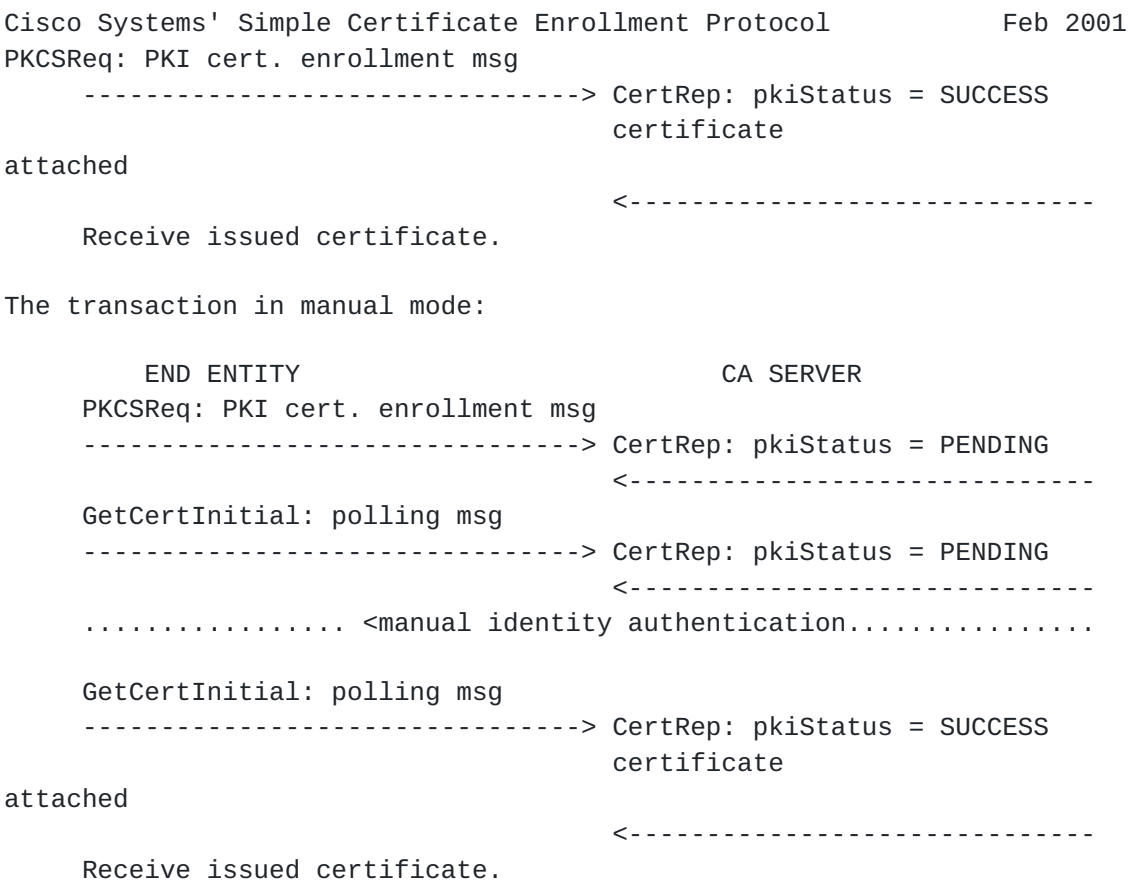
2.2.3 Certificate Enrollment

An end entity starts an enrollment transaction by creating a certificate request using PKCS#10 and send it to the CA/RA enveloped using the PKCS#7. After the CA/RA receives the request, it will either automatically approve the request and send the certificate back, or it will require the end entity to wait until the operator can manually authenticate the identity of the requesting end entity. Two attributes (defined in PKCS#6) are included in the PKCS#10 certificate request - a Challenge Password attribute and an optional ExtensionReq attribute which will be a sequence of extensions the end entity would like to be included in its V3 certificate extensions. The Challenge Password is used for revocation and may be used (at the option of the CA/RA) additionally as a one-time password for automatic enrollment.

In the automatic mode, the transaction consists of one PKCSReq PKI Message, and one CertRep PKI message. In the manual mode, the end entity enters into polling mode by periodically sending GetCertInitial PKI message to the server, until the server operator completes the manual authentication, after which the CA will respond to GetCertInitial by returning the issued certificate.

The transaction in automatic mode:

END ENTITY CA SERVER



2.2.4 End Entity Certificate Revocation

An end entity should be able to revoke its own certificate. Currently the revocation is implemented as a manual process. In order to revoke a certificate, the end entity make a phone call to the CA server operator. The operator will come back asking the ChallangePassword (which has been send to the server as an attribute of the PKCS#10 certificate request). If the ChallangePassword matches, the certificate is revoked. The reason of the revocation is documented by CA/RA.

2.2.5 Certificate Access

There are two methods to query certificates. The first method is to use LDAP as a query protocol. Using LDAP to query assumes the client understand the LDAP scheme supported by the CA. The SCEP client assumes that the subject DN name in the certificate is used as URL to query the certificate. The standard attributes (userCertificate and caCertificate) are used as filter.

For the environment where LDAP is not available, a certificate query message is defined to retrieve the certificates from CA.

To query a certificate from the certificate authority, an end entity sends a request consisting of the certificate's issuer name and the

serial number. This assumes that the end entity has saved the issuer

name and the serial number of the issued certificate from the previous enrollment transaction. The transaction to query a certificate consists of one GetCert PKI message and one CertRep PKI message:

```
         END ENTITY                          CA SERVER
     GetCert: PKI cert query msg
     ------------------------------> CertRep:  pkiStatus = SUCCESS
                                     certificate
attached
                                     <----------------------------
     Receive the certificate.
```

2.2.6 CRL Distribution

The CA/RA will not "push" the CRL to the end entities. The query of the CRL can only be initialized by the end entity.

There are three methods to query CRL.

The CRL may be retrieved by a simple HTTP GET. If the CA supports this method, it should encode the URL into a CRL Distribution Point extension in the certificates it issues. Support for this method should be incorporated in new and updated clients, but may not be in older versions.

The second method is to query CRL using LDAP. This assumes the CA server supports CRL LDAP publishing and issues the CRL Distribution Point in the certificate. The CRL Distribution Point is encoded as a DN. Please refer to Appendix D for the examples of CRL Distribution Point.

The third method is implemented for the CA which does not support LDAP CRL publishing or does not implement the CRL Distribution Point. In this case, a CRL query is composed by creating a message consists of the CA issuer name and the CA's certificate serial number. This method is deprecated because it does not scale well and requires the CA to be a high-availability service.

The message is send to the CA in the same way as the other SCEP requests: The transaction to query CRL consists of one GetCRL PKI message and one CertRep PKI message which have no certificates but CRL.

```
         END ENTITY                          CA SERVER
     GetCRL: PKI CRL query msg
     ---------------------------------> CertRep:  CRL attached
                                       <--------------------------------
```

2.3 PKI Operation Transactional Behavior

As described before, a PKI operation is a transaction consisting of the messages exchanged between an end entity and the CA/RA. This section

will specify the transaction behavior on both the end entity and the

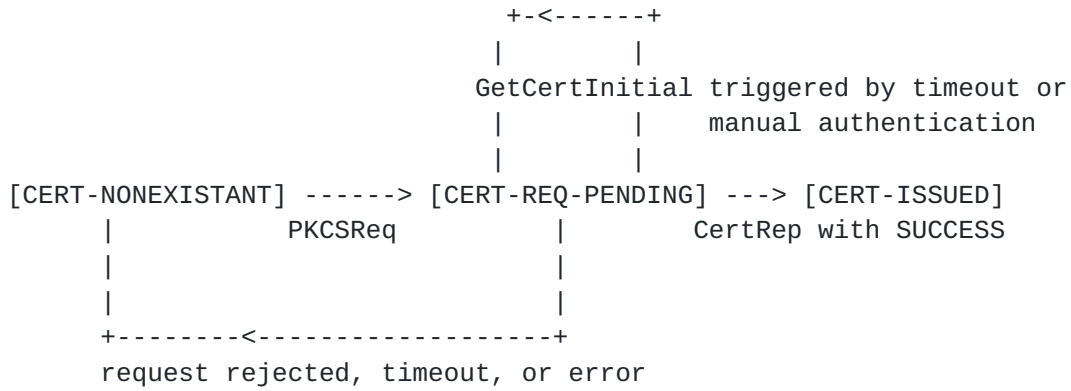
certificate authority server. Because the protocol is basically a two way communication protocol without a confirmation message from the initiating side, state and state resynchronization rules have to be defined, in case any error happens at either side. Before the state transition can be defined, the notion of transaction identifier has to be defined first.

2.3.1 Transaction Identifier

A transaction identifier is a string generated by the entity when starting a transaction. Since all the PKI operations defined in this protocol are initiated by the end entity, it is the responsibility of the end entity to generate a unique string as the transaction identifier. All the PKI messages exchanged for a given PKI operations must carry the same transaction identifier. The transaction identifier is generated as a MD5 hash on the public key value for which the enrollment request is made. This allows the SCEP client to reuse the same transaction identifier if it is reissuing a request for the same certificate (i.e. a certificate with the same subject, issuer, and key). The SCEP protocol requires that transaction identifiers be unique, so that queries can be matched up with transactions. For this reason, in those cases in which separate signing and encryption certificates are issued to the same end entity, the keys must be different.

2.3.2 State Transitions in Certificate Enrollment

The end entity state transitions during enrollment operation is indicated in the diagram below:



As described in the section 2.2.3, certificate enrollment starts at the state CERT-NONEXISTANT. Sending PKCSReq changes the state to CERT-REQ-PENDING. Receiving CertRep with SUCCESS status changes the state to CERT-ISSUED. In the case the server sending back the response with pending status, the end entity will keep polling certificate response by sending GetCertInitial to the server, until either a CertRep with SUCCESS status is received, or the maximum polling number has been exceeded.

If an error or timeout occurs in the CERT-REQ-PENDING state, the end entity will transition to the CERT-NONEXISTANT state.

The client administrator will, eventually, start up another enrollment request. It is important to note that, as long as the end entity does not change its subject name or keys, the same transaction id will be used in the "new" transaction. This is important because based on this transaction id, the certificate authority server can recognize this as an existing transaction instead of a new one.

2.3.3 Transaction Behavior of Certificate/CRL Access

There is no state maintained during certificate access and CRL access transaction. When using the certificate query and CRL query messages defined in this protocol, the transaction identifier is still required so that the end entity can match the response message with the upstanding request message. When using LDAP to query the certificate and the CRL, the behavior is specified by the LDAP protocol.

2.4 Security

The security goals of SCEP are that no adversary can:

- subvert the public key/identity binding from that intended,
- discover the identity information in the enrollment requests and issued certificates,
- cause the revocation of certificates with any non-negligible probability.

Here an adversary is any entity other than the end entity and the CA (and optionally the RA) participating in the protocol that is computationally limited, but that can manipulate data during transmission (that is, a man-in-the-middle). The precise meaning of 'computationally limited' depends on the implementer's choice of cryptographic hash functions and ciphers. The required algorithms are RSA, DES, and MD5.

The first and second goals are met through the use of PKCS#7 and PKCS#10 encryption and digital signatures using authenticated public keys. The CA's public key is authenticated via the checking of the CA fingerprint, as specified in Section 2.1.2, and the SCEP client's public key is authenticated through the manual authentication or pre-shared secret authentication, as specified in Section 2.1.1.2. The third goal is met through the use of a Challenge Password for revocation, that is chosen by the SCEP client and communicated to the CA protected by the PKCS#7 encryption, as specified in Section 2.2.4.

The motivation of the first security goal is straightforward. The motivation for the second security goal is to protect the identity information in the enrollment requests and certificates. For example, two IPSEC hosts behind a firewall may need to exchange certificates, and

may need to enroll certificates with a CA that is outside of a firewall. Most networks with firewalls seek to prevent IP addresses and DNS

information from the trusted network leaving that network. The second goal enables the hosts in this example to enroll with a CA outside the firewall without revealing this information. The motivation for the third security goal is to protect the SCEP clients from denial of service attacks.

Section 3 Transport Protocol

In the SCEP protocol, HTTP is used as the transport protocol for the PKI messages.

3.1 HTTP "GET" Message Format

In the PKI protocol, CA/RA certificates are send to the end entity in clear, whereas the end entity certificates are send out using the PKCS#7 secure protocol. This results in two types of GET operations. The type of GET operation is specified by augmenting the GET message with OPERATION and MESSAGE parameters in the Request-URL. OPERATION identifies the type of GET operation, and MESSAGE is actually the PKI message encoded as a text string.

The following is the syntax definition of a HTTP GET message send from an end entity to a certificate authority server:

```
Request = "GET " CGI-PATH CGI-PROG "?operation=" OPERATION "&message=" MESSAGE
where:
  CGI-PATH defines the actual CGI path to invoke the CGI program which
  parses the request.
|  CGI-PROG is set to be the string "pkiclient.exe".  This is intended
|  to be the program that the CA will use to handle the SCEP transactions,
|  though the CA may ignore CGI-PROG and use only the CGI-PATH.
  OPERATION is set to be the string "PKIOperation" when the GET message
  carries a PKI message to request certificates or CRL; OPERATION is set
|  to be the string "GetCACert" or "GetCACertChain" when the GET operation
|  is used to get CA/RA certificate or the CA Cert chain (respectively).
  When OPERATION is "PKIOperation", MESSAGE is a base64-encoded PKI
  message
|  when OPERATION is "GetCACert" or "GetCACertChain", MESSAGE is a string
|  which represents the certificate authority issuer identifier.
```

For example. An end entity may submit a message via HTTP to the server as follows:

```
GET /cgi-bin/pkiclient.exe?operation=PKIOperation&message=MIAGCSqGSIb3D
QEHA6CAMIACAQAxgDCBzAIBADB2MGIxETAPBgNVBAcTCE ......AAAAAA==
```

3.2 Response Message Format

For each GET operation, the CA/RA server will return a MIME object via

HTTP. For a GET operation with PKIOperation as its type, the response is tagged as having a Content Type of application/x-pki-message. The body of this message is a BER encoded binary PKI message. The following is an example of the response:

"Content-Type:application/x-pki-message\n\n"<BER-encoded PKI msg>

In the case of GET operation with a type of GetCACert, the MIME content type returned will depend on whether or not an RA is in use. If there is no RA, only the CA certificate is send back in the response, and the response has the content type tagged as application/x-x509-ca-cert. the body of the response is a DER encoded binary X.509 certificate. For example:

"Content-Type:application/x-x509-ca-cert\n\n"<BER-encoded X509>

If there is an RA, the RA certificates are send back together with the CA certificates, a certificate-only PKCS#7 SignedData is send back in the response where the SignerInfo is empty. Section 5 has the detailed definition of the message format in this case. The content type is application/x-x509-ca-ra-cert.

Section 4 Secure Transportation: PKCS#7

PKCS#7 is a general enveloping mechanism that enables both signed and encrypted transmission of arbitrary data. It is widely implemented and included in the RSA tool kit.

In this section, the general PKCS#7 enveloped PKI message format is specified. The complete PKCS#7 message format for each PKI transaction will be covered in Section 5.

4.1 SCEP Message Format

As a transaction message, a SCEP message has a set of transaction specific attributes and an information portion. Employing PKCS#7 protocol, the transaction specific attributes are encoded as a set of authenticated attributes of the SignedData. The information portion will first be encrypted to become Enveloped Data, and then the digest of the enveloped information portion is included as one of the message digest attributes and being signed together with the other transaction specific attributes.

By applying both enveloping and signing transformations, a SCEP message is protected both for the integrity of its end-end-transition information and the confidentiality of its information portion. The advantage of this technique over the conventional transaction message format is that, the signed transaction type information and the status of the transaction can be determined prior to invoke security handling procedures specific to the information portion being processed.

The following is an example of a SCEP message with its enveloped and signed data portion represented by pkcsPKISigned and pkcsPKIEnveloped. The out-most of any PKI message is a blob of ContentInfo, with its content type set to SignedData and the actual

signed data as the content.

```
  pkiMessage ContentInfo ::= {
      contentType {pkcs-7 signedData(2)}
      content pkcsPKISigned
  }
  pkcsPKISigned SignedData ::= {
      version 1
      digestAlgorithm { iso(1) member-body(2) US(840) rsadsi(113549)
                        digestAlgorithm(2) 5}
      contentInfo {
          contentType {pkcs-7 1} --  data content identifier
          content pkcsPKIEnvelope -- enveloped information portion
      }
      certificates   -- signer certificate chain
      signerInfo     -- including signed transaction info and the digest
                     -- of the enveloped information portion as the
                     -- authenticated attributes
  }
  pkcsPKIEnveloped EnvelopedData ::= {
      version 0
      recipientInfos -- information required to open the envelop
      encryptedContentInfo {
          contentType {pkcs-7 1}  -- data content identifier
          contentEncryptionAlgorithm
          encryptedContent     -- encrypted information portion
      }
  }
```

4.2 Signed Transaction Attributes

The following transaction attributes are encoded as authenticated attributes. Please refer to Appendix B for the OID definitions.

```
transactionID      PrintableString  -- Decimal value as a string
  messageType        PrintableString  -- Decimal value as a string
  pkiStatus          PrintableString  -- Decimal value as a string
  failinfo           PrintableString  -- Decimal value as a string
  senderNonce        Octet String
  recipientNonce     Octet String
```

where:

The transactionID is an attribute which uniquely identify a transaction. This attribute is required in all PKI messages.

The messageType attribute specify the type of operation performed by the transaction. This attribute is required in all PKI messages. Currently, the following message types are defined:

```
    PKCSReq (19)  -- Permits use of PKCS#10 certificate request
```

```
    CertRep (3)   -- Response to certificate or CRL request
    GetCertInitial (20)  -- Certificate polling in manual enrollment
    GetCert (21)  -- Retrieve a certificate
```

```
    GetCRL  (22)  -- Retrieve a CRL
```

All response message will include transaction status information which is defined as pkiStatus attribute:

```
        SUCCESS (0)   -- request granted
        FAILURE (2)   -- request rejected
        PENDING (3)   -- request pending for manual approval.
```

If the status in the response is FAILURE, the failinfo attribute will contain one of the following failure reasons:

```
      badAlg (0)  -- Unrecognized or unsupported algorithm ident
      badMessageCheck (1)  -- integrity check failed
      badRequest (2)  -- transaction not permitted or supported
      badTime (3)  -- Message time field was not sufficiently close
                      to the system time
      badCertId (4)  -- No certificate could be identified matching
                        the provided criteria
```

The attributes of senderNonce and recipientNonce are the 16 byte random numbers generated for each transaction to prevent the replay attack.

When an end entity sends a PKI message to the server, a senderNonce is included in the message. After the server processes the request, it will send back the end entity senderNonce as the recipientNonce and generates another nonce as the senderNonce in the response message. Because the proposed pki protocol is a two-way communication protocol, it is clear that the nonce can only be used by the end entity to prevent the replay. The server has to employ extra state related information to prevent a replay attack.

Section 5. SCEP Transaction Specification

In this section each SCEP transaction is specified in terms of the complete messages exchanged during the transaction.

5.1 Certificate Enrollment

The certificate enrollment transaction consists of one PKCSReq message send to the certificate authority from an end entity, and one CertRep message send back from the server. The pkiStatus returned in the response message is either SUCCESS, or FAILURE, or PENDING. The information portion of a PKCSReq message is a PKCS#10 certificate request, which contains the subject Distinguished Name, the subject public key, and two attributes, a ChallangePassword attribute to be used for revocation, and an optional ExtensionReq attribute which will be a sequence of extensions the end entity expects to be included in its V3 certificate extensions. One of the extension attribute specifies the key

usage. The pkiStatus is set to SUCCESS when the certificate is send back in CertRep; the pkiStatus is set to FAILURE when the certificate

request is rejected; the pkiStatus is set to PENDING when the server has decided to manually authenticate the end entity. The messages exchanged in the manual authentication mode is further specified in Section 5.2.

Precondition:
Both the end entity and the certificate authority have completed their initialization process. The end entity has already been configured with the CA/RA certificate.

Postcondition:
Either the certificate is received by the end entity, or the end entity is notified to do the manual authentication, or the request is rejected.

5.1.1 PKCSReq Message Format

A PKCSReq message is created by following the steps defined below:

1. **Create a PKCS#10 certificate request which is signed by the end entity's private key, corresponding to the public key included in** the PKCS#10 certificate request. This constitutes the information portion of PKCSReq.

2. **Encrypt the PKCS#10 certificate request using a randomly generated content-encryption key. This content-encryption key is then** encrypted by the CA's* public key and included in the recipientInfo. This step completes the "envelope" for the PKCS#10 certificate request.

3. **Generate a unique string as the transaction id.**

4. **Generate a 16 byte random number as senderNonce.**

5. **Generate message digest on the enveloped PKCS#10 certificate request using the selected digest algorithm.**

6. **Create SignedData by adding the end entity's self-signed certificate as the signer's public key certificate. Include the message type,** transaction id, the senderNonce and the message digest as the authenticated attributes and sign the attributes using the end entity's private key. This completes the SignedData.

7. **The SignedData is prepended with the ContenInfo blob which indicates a SignedData object. This final step completes the create of a** complete PKCSReq PKI message.

In the following, the PKCSReq message is defined following the ASN.1 notation.

For readability, the values of a field is either represented by a quoted

string which specifies the intended value, or a constant when the value is known.

```
  -- PKCSReq information portion
  pkcsCertReq CertificationRequest ::= {   -- PKCS#10
      version 0
      subject  "the end entity's subject name"
      subjectPublicKeyInfo {
          algorithm {pkcs-1 1}  -- rsa encryption
          subjectPublicKey "DER encoding of the end entity's public key"
      }
      attributes {
          challengePassword {{pkcs-9 7} "password string" }
          extensions
      }
      signatureAlgorithm {pkcs-1 4} -- MD5WithRSAEncryption
      signature "bit string which is created by signing inner content
                 of the defined pkcsCertReq using end entity's private
                 key, corresponding to the public key included in
                 subjectPublicKeyInfo."
  }
  -- Enveloped information portion
  pkcsCertReqEnvelope EnvelopeData ::= {   -- PKCS#7
      version 0
      recipientInfo {
          version 0
          issuerAndSerialNumber {
              issuer  "the CA issuer name"
              serialNumber  "the CA certificate serial number"
          }
          keyEncryptionAlgorithm  {pkcs-1 1}  -- rsa encryption
          encryptedKey "content-encryption key
                        encrypted by CA public key"
      }
      encryptedContentInfo {
          contentType {pkcs-7 1}  -- data content
          contentEncryptionAlgorithm  "object identifier
                                       for DES encryption"
          encryptedContent  "encrypted pkcsCertReq using the content-
                             encryption key"
      }
  }
  -- Signed PKCSReq
  pkcsCertReqSigned SignedData ::= { -- PKCS#7
      version 1
      digestAlgorithm {iso(1) member-body(2) US(840) rsadsi(113549)
                       digestAlgorithm(2) 5}
      contentInfo {
          contentType {pkcs-7 1} -- data content identifier
          content  pkcsCertReqEnvelope
      }
      certificate {   -- the end entity's self-signed certificate
```

```
version 3
serialNumber  "the transaction id associated with enrollment"
signature {pkcs-1 4}  -- md5WithRSAEncryption
```

```
     issuer " the end entity's subject name"
          validity {
              notBefore "a UTC time"
              notAfter  "a UTC time"
          }
          subject  "the end entity's subject name"
          subjectPublicKeyInfo {
              algorithm {pkcs-1 1}
              subjectPublicKey "DER encoding of end entity's public key"
          }
          signatureAlgorithm {pkcs-1 4}
          signature "the signature generated by using the end entity's
                     private key corresponding to the public key in
                     this certificate."
      }
      signerInfo  {
          version 1
          issuerAndSerialNumber {
              issuer "the end entity's subject name"
              serialNumber "the transaction id associated
                            with the enrollment"
          }
          digestAlgorithm {iso(0) member-body(2) US(840) rsadsi(113549)
                           digestAlgorithm(2) 5}
          authenticateAttributes {
              contentType  {{pkcs-9 3} {pkcs-7 1}}
              messageDigest {{pkcs-9 4} "an octet string"}
              transaction-id {{id-attributes transId(7)} "printable
                                                          string"}
                             -- this transaction id will be used
                             -- together with the subject name as
                             -- the identifier of the end entity's key
                             -- pair during enrollment
              messageType {{id-attributes messageType(2)} "PKCSReq"}
              senderNonce {{id-attributes senderNonce(5)}
                           "a random number encoded as a string"}
          }
          digestEncryptionAlgorithm {pkcs-1 1} -- rsa encryption
          encryptedDigest "encrypted digest of the authenticated
                           attributes using end entity's private key"
      }
  }
  pkcsReq PKIMessage ::= {
      contentType {pkcs-7 2}
      content pkcsCertRepSigned
  }
```

5.1.2 CertRep Message Format

The response to an SCEP enrollment request is a CertRep message.

5.1.2.1 PENDING Response

When the CA is configured to manually authenticate the end entity, the CertRep is returned with the attribute pkiStatus set to PENDING. The data portion for this message is null. Only the transaction required attributes are sent back.

```
CertRepSigned  SignedData ::= { -- PKCS#7
    version 1
    digestAlgorithm {iso(1) member-body(2) US(840) rsadsi(113549)
                     digestAlgorithm(2) 5}
    contentInfo {contentType {pkcs-7 1}  -- empty content
    }
    signerInfo {
        version 1
        issuerAndSerialNumber {
|            issuer "name of CA that issued the CA [RA] cert"
|            serialNumber "the serial number of the CA [RA] cert"
        }
        digestAlgorithm (iso(1) member-body(2) US(840) rsadsi(113549)
                         digestAlgorithm(2) 5}
        authenticateAttributes {
            contentType {{pkcs-9 3} {pkcs-7 1}}
            messageDigest {{pkcs-9 4} NULL}
            messageType {{id-attribute messageType(0)} "CertRep"}
            transaction-id {{id-attributes transid(7)} "printablestring"}
                           --- same transaction id used in PKCSReq
            pkiStatus  {{id-attributes pkiStatus(3)} "PENDING"}
            recipientNonce {{id-attributes recipientNonce(6)}<16 bytes>}
            senderNonce {{id-attributes senderNonce(5)} <16 bytes>}
        }
        digestEncrytionAlgorithm {pkcs-1 1}
        encryptedDigest  "encrypted message digest of the authenticated
|                          attributes using the CA's [RA's] private key"
     }
}
CertRep PKIMessage ::= {
    contentType {pkcs-7 2}
    content CertRepSigned
}
```

5.1.2.2 Failure Response

In this case, the CertRep sent back to the end entity is same as in the PENDING case, except that the pkiStatus attribute is set to FAILURE,

and the failInfo attribute should be included:

```
    pkistatus {{id-attributes pkiStatus(3)} "FAILURE"}
    failInfo {{id-attributes failInfo(4)} "the reason to reject"}
```

5.1.2.3 SUCCESS response

In this case, the information portion of CertRep will be a degenerated PKCS#7 which contains the end entity's certificate. It is then enveloped and signed as below:

```
pkcsCertRep SignedData ::= { -- PKCS#7
    version 1
    digestAlgorithm {iso(1) member-body(2) US(840) rsadsi(113549)
                     digestAlgorithm(2) 5}
    contentInfo { -- empty content since this is degenerated PKCS#7
        contentType {pkcs-7 1}
    }
    certificates {
        certificate { -- issued end entity's certificate
            version 3
            serialNumber "issued end entity's certificate serial number"
            signature {pkcs-1 4}  -- md5WithRSAEncryption
            issuer "the certificate authority issuer name"
            validity {
                notBefore "UTC time"
                notAfter "UTC time"
            }
            subject  "the end entity subject name as given in PKCS#10"
            subjectPublicKeyInfo {
                algorithm {pkcs-1 1}
                subjectPublicKey "a DER encoding of end entity public
                                  key as given in PKCS#10"
            }
            extensions  " the extensions as given in PKCS#10"
            signatureAlgorithm {pkcs-1 4}
            signature " the certificate authority signature"
        }
        certificate "the certificate authority certificate"
|        certificate "the registration authority certificate (if applicable)"
    }
}
pkcsCertRepEnvelope EnvelopedData ::= {  -- PKCS#7
    version 0
    recipientInfo {
        version 0
        issuerAndSerialNumber { -- use issuer name and serial number as
                                -- conveyed in end entity's self-signed
                                -- certificate, included in the PKCSReq
            issuer  "the end entity's subject name"
            serialNumber "the serial number defined by the end entity in
                          its self-signed certificate"
        }
        keyEncryptionAlgorithm {pkcs-1 1}
```

```
        encryptedKey "content-encrypt key encrypted by the end entity's
                      public key which is same key as authenticated in
                      the end entity's certificate"
    }
```

```
    encryptedContentInfo {
        contentType {pkcs-7 1}  -- data content identifier
        contentEncryptionAlgorithm "OID for DES encryption"
        encryptedContent "encrypted pkcsCertRep using content encryption
                          key"
    }
}
pkcsCertRepSigned SignedData ::= { -- PKCS#7
    version 1
    digestAlgorithm {iso(1) member-body(2) US(840) rsadsi(113549)
                     digestAlgorithm(2) 5}
    contentInfo {
        contentType {pkcs-7 1}
        content pkcsCertRepEnvelope
    }
    signerInfo {
        version 1
        issuerAndSerialNumber {
            issuer "the certificate authority issuer name"
            serialNumber "the CA certificate's serial number"
        }
        digestAlgorithm {iso(1), member-body(2) US(840) rsadsi(113549)
                         digestAlgorithm(2) 5}
        authenticateAttributes {
            contentType {{pkcs-9 3} {pkcs-7 1}}
            messageDigest {{pkcs-9 4} "a octet string"}
            messageType {{id-attribute messageType(2)} "CertRep"}
            transaction-id {{id-attributes transId(7)} "printable
                                                        string"}
                          -- same transaction id as given in PKCSReq
            pkiStatus {{id-attributes pkiStatus(3) "SUCCESS"}
            recipientNonce {{id-attribute recipientNonce(6)}<16 bytes>}
            senderNonce {{ id-attributes senderNonce(5) <16 bytes>}
        }
        digestEncryptionAlgorithm {pkcs-1 1}
        encryptedDigest "encrypted digest of authenticate attributes
                         using CA's private key "
    }
}
CertRep PKIMessage ::= {
    contentType {pkcs-7 2}
    content pkcsCertRepSigned
}
```

5.2 Poll for End Entity Initial Certificate

Either triggered by the PENDING status received from the CertRep, or by the non-response timeout for the previous PKCSReq, an end entity will enter the polling state by periodically sending GetCertInitial to the

server, until either the request is granted and the certificate is sent back, or the request is rejected, or the the configured time limit for polling is exceeded.

Since GetCertInitial is part of the enrollment, the messages exchanged during the polling period should carries the same transaction identifier as the previous PKCSReq.

PreCondition
    Either the end entity has received a CertRep with pkiStatus set to be PENDING, or the previous PKCSReq has timed out.

PostContition
    The end entity has either received the certificate, or be rejected of its request, or the polling period ended as a failure.

5.2.1 GetCertInitial Message Format

|Since at this time the certificate has not been issued, the end entity
|can only use the end entity's subject name, combined with the
|transaction identifier, to identify the polled certificate request.

|The certificate authority server must be able to uniquely identify the
|polled certificate request. A subject name can have more than one
|outstanding certificate request (with different key usage attributes).

```
-- Information portion

pkcsGetCertInitial issuerAndSubject ::= {
    issuer "the certificate authority issuer name"
    subject "the end entity subject name as given in PKCS#10"
}
pkcsGetCertInitialEnvelope EnvelopedData ::= {
    version 0
    recipientInfo {
        version 0
        issuerAndSerialNumber {
            issuer  "the CA issuer name"
            serialNumber "the CA certificate serial number"
        }
        keyEncryptionAlgorithm {pkcs-1 1}
        encryptedKey "content-encrypt key encrypted by CA's public key"
    }
    encryptedContentInfo {
        contentType {pkcs-7 1}  -- data content
        contentEncryptionAlgorithm "OID for DES encryption"
        encryptedContent "encrypted getCertInital"
    }
}
pkcsGetCertInitialSigned SignedData ::= { -- PKCS#7
    version 1
    digestAlgorithm {iso(1) member-body(2) US(840) rsadsi(113549)
```

```
                    digestAlgorithm(2) 5}
    contentInfo {
        contentType {pkcs-7 1}
```

```
        content pkcsGetCertIntialEnvelope
    }
     certificate {   -- the end entity's self-signed certificate
         version 3
         serialNumber  "the transaction id associated with enrollment"
         signature {pkcs-1 4}  -- md5WithRSAEncryption
         issuer " the end entity's subject name"
         validity {
             notBefore "a UTC time"
             notAfter  "a UTC time"
         }
         subject  "the end entity's subject name"
         subjectPublicKeyInfo {
             algorithm {pkcs-1 1}
             subjectPublicKey "DER encoding of end entity's public key"
         }
         signatureAlgorithm {pkcs-1 4}
         signature "the signature generated by using the end entity's
                    private key corresponding to the public key in
                    this certificate."
     }
    signerInfo {
        version 1
        issuerAndSerialNumber {
            issuer "end entity's subject name"
            serialNumber "the transaction id used in previous PKCSReq"
        }
        digestAlgorithm {iso(1), member-body(2) US(840) rsadsi(113549)
                         digestAlgorithm(2) 5}
        authenticateAttributes {
            contentType {{pkcs-9 3} {pkcs-7 1}}
            messageDigest {{pkcs-9 4} "an octet string"}
                           -- digest of getCertInitial
            messageType {{id-attribute messageType(2)} "GetCertInitial"}
            transaction-id {{id-attributes transId(7)} "printable
                                                        string"}
                          -- same transaction idused in previous PKCSReq
            senderNonce {{id-attribute senderNonce(3)} 0x<16 bytes>}
        }
        digestEncryptionAlgorithm {pkcs-1 1}
        encryptedDigest "encrypted digest of authenticateAttributes"
    }
}
GetCertInitial PKIMessage ::= {
    contentType {pkcs-7 2}
    content pkcsGetCertInitialSigned
}
```

5.2.2 GetCertInitial Response Message Format

The response messages for GetCertInitial are the same as for PKCSReq.

5.3 Certificate Access

The certificate query message defined in this section is an option when the LDAP server is not available to provide the certificate query. An end entity should be able to query an issued certificate from the certificate authority, as long as the issuer name and the issuer assigned certificate serial number is known to the requesting end entity. This transaction is not intended to provide the service as a certificate directory service. A more complicated query mechanism would have to be defined in order to allow an end entity to query a certificate using various different fields.

This transaction consists of one GetCert message send to the server by an end entity, and one CertRep message send back from the server.

PreCondition
The queried certificate have been issued by the certificate authority and the issuer assigned serial number is known.

PostContition
Either the certificate is send back or the request is rejected.

5.3.1 GetCert Message Format

The queried certificate is identified by its issuer name and the issuer assigned serial number. If this is a query for an arbitrary end entity's certificate, the requesting end entity should includes its own CA issued certificate in the signed envelope. If this is a query for its own certificate (assume the end entity lost the issued certificate, or does not have enough non-volatile memory to save the certificate), then the self-signed certificate has to be included in the signed envelope.

```
  pkcsGetCert issuerAndSerialNumber ::= {
      issuer "the certificate issuer name"
      serialNumber "the certificate serial number"
  }
  pkcsGetCertEnvelope EnvelopedData ::= {
      version 0
      recipientInfo {
          version 0
          issuerAndSerialNumber {
              issuer  "the CA [RA] issuer name"
              serialNumber "the CA [RA] certificate serial number"
          }
          keyEncryptionAlgorithm {pkcs-1 1}
          encryptedKey "content-encrypt key encrypted
                        by CA [RA] public key"
      }
```

```
      encryptedContentInfo {
          contentType {pkcs-7 1}  -- data content
          contentEncryptionAlgorithm "OID for DES encryption"
          encryptedContent "encrypted pkcsGetCert using the content
                            encryption key"
      }
  }
  pkcsGetCertSigned SignedData ::= {
      version 1
      digestAlgorithm {iso(1) member-body(2) US(840) rsadsi(113549)
                       digestAlgorithm(2) 5}
      contentInfo {
          contentType {pkcs-7 1}
          content pkcsGetCertEnvelope
      }
      certificates {
          certificate "CA issued certificate"
                      or "self-signed certificate"
      }
      signerInfo {
          version 1
          issuerAndSerialNumber {
              issuer "the end entity's subject name"
              serialNumber "end entity's certificate serial number"
          }
          digestAlgorithm {iso(1), member-body(2) US(840) rsadsi(113549)
                           digestAlgorithm(2) 5}
          authenticateAttributes {
              contentType {{pkcs-9 3} {pkcs-7 1}}
              messageDigest {{pkcs-9 4} "an octet string"}
                             -- digest of pkcsGetCertEnvelope
              messageType {{id-attribute messageType(2)} "GetCert"}
              transaction-id {{id-attributes transId(7)} "printable
                                                          string"}
              senderNonce {{id-attribute senderNonce(3)} <16 bytes>}
          }
          digestEncryptionAlgorithm {pkcs-1 1}
          encryptedDigest "encrypted digest of authenticateAttributes"
      }
  }
  GetCert PKIMessage ::= {
      contentType {pkcs-7 2}
      content pkcsGetCertSigned
  }
```

5.3.2 CertRep Message Format

In this case, the CertRep from the server is same as the CertRep for the PKCSReq, except that the server will only either grant the request or reject the request. Also, the recipientInfo should use the CA issuer name and CA assigned serial number to identify the end entity's key pair since at this time, the end entity has received its own certificate.

5.4 CRL Access

The CRL query message defined in this section is an option when the LDAP server is not available to provide the CRL query. In the PKI protocol proposed here, only the end entity can initiate the transaction to download CRL. An end entity send GetCRL request to the server and the server send back CertRep whose information portion is a degenerated PKCS#7 which contains only the most recent CRL. The size of CRL included in the CertRep should be determined by the implementation.

PreCondition
    The certificate authority certificate has been downloaded to the end entity.

PostCondition
    CRL send back to the end entity.

5.4.1 GetCRL Message format

The CRL is identified by using both CA's issuer name and the CA certificate's serial number:

```
  pkcsGetCRL issuerAndSerialNumber {
      issuer "the certificate authority issuer name"
      serialNumber "certificate authority certificate's serial number"
  }
```

When the CRLDistributionPoint is supported, the pkcsGetCRL is defined as the following:

```
  pkcsGetCRL SEQUENCE {
      crlIssuer  issuerAndSerialNumber
      distributionPoint CE-CRLDistPoints
  }
```

where CE-CRLDisPoints is defined in X.509, but must contain only one CRL distribution point.

```
  pkcsGetCRLEnvelope EnvelopedData ::= {
      version 0
      recipientInfo {
          version 0
          issuerAndSerialNumber {
              issuer  "the certificate authority (or RA) issuer name"
              serialNumber "the CA (RA) certificate's serial number"
          }
          keyEncryptionAlgorithm {pkcs-1 1}
          encryptedKey "content-encrypt key encrypted by CA (RA) public key"
      }
      encryptedContentInfo {
          contentType {pkcs-7 1}  -- data content
          contentEncryptionAlgorithm "OID for DES encryption"
          encryptedContent "encrypted pkcsGetCRL"
      }
  }
  pkcsGetCRLSigned SignedData ::= {
      version 1
      digestAlgorithm {iso(1) member-body(2) US(840) rsadsi(113549)
                       digestAlgorithm(2) 5}
      contentInfo {
          contentType {pkcs-7 1}
          content pkcsGetCRLEnvelope
      }
      certificates {
         certificate "CA-issued or self-signed end entity's certificate"
      }
      signerInfo {
          version 1
          issuerAndSerialNumber {
              issuer "the end entity's issuer name"
              serialNumber "the end entity's certificate serial number"
          }
          digestAlgorithm {iso(1), member-body(2) US(840) rsadsi(113549)
                           digestAlgorithm(2) 5}
          authenticateAttributes {
              contentType {{pkcs-9 3} {pkcs-7 1}}
              messageDigest {{pkcs-9 4} 0x<16/20 bytes>}
                              -- digest of pkcsGetCRLEnvelope
              messageType {{id-attribute messageType(2)} "CertCRL"}
              transaction-id {{id-attributes transId(7)} "printable
                                                          string"}
              senderNonce {{id-attribute senderNonce(3)} <16 bytes>}
          }
          digestEncryptionAlgorithm {pkcs-1 1}
          encryptedDigest "encrypted digest of authenticateAttributes"
      }
  }
```

```
  GetCRL PKIMessage ::= {
      contentType {pkcs-7 2}
      content pkcsGetCRLSigned
  }
```

5.4.2 CertRep Message Format

The CRL is send back to the end entity through CertRep message. The information portion of this message is a degenerated PKCS#7 SignedData which contains only a CRL.

```
  pkcsCertRep SignedData ::= {
      version 1
      digestAlgorithm {iso(1) member-body(2) US(840) rsadsi(113549)
                       digestAlgorithm(2) 5}
      contentInfo {
          contentType {pkcs-7 1}
      }
      crl {
          signature {pkcs-1 4}
          issuer "the certificate authority issuer name"
          lastUpdate "UTC time"
          nextUpdate "UTC time"
          revokedCertificate {
              -- the first entry
              userCertificate "certificate serial number"
              revocationData "UTC time"
              ....
              -- last entry
              userCertificate "certificate serial number"
              revocationData "UTC time"
      }
  }
  pkcsCertRepEnvelope EnvelopedData ::= {
      version 0
      recipientInfo {
          version 0
          issuerAndSerialNumber {
              issuer  "the end entity's issuer name"
              serialNumber "the end entity certificate serial number"
          }
          keyEncryptionAlgorithm {pkcs-1 1}
          encryptedKey "content-encrypt key encrypted by end entity's
                        public key "
      }
      encryptedContentInfo {
          contentType {pkcs-7 1}  -- data content
          contentEncryptionAlgorithm "OID for DES encryption"
          encryptedContent "encrypted pkcsCertRep using end entity's
                            public key"
      }
  }
```

```
  pkcsCertRepSigned SignedData ::= { -- PKCS#7
      version 1
      digestAlgorithm {iso(1) member-body(2) US(840) rsadsi(113549)
                       digestAlgorithm(2) 5}
      contentInfo {
          contentType {pkcs-7 1}
          content pkcsCertRepEnvelope
      }
      signerInfo {
          version 1
          issuerAndSerialNumber {
              issuer "the certificate authority issuer name"
              serialNumber "the CA certificate's serial number"
          }
          digestAlgorithm {iso(1), member-body(2) US(840) rsadsi(113549)
                           digestAlgorithm(2) 5}
          authenticateAttributes {
              contentType {{pkcs-9 3} {pkcs-7 1}}
              messageDigest {{pkcs-9 4} "an octet string"}
                            -- digest of pkcsCertRepEnvelope
              messageType {{id-attribute messageType(2)} "CertRep"}
              transaction-id {{id-attributes transId(7)} "printable
                                                          string"}
                            -- same transaction id as given in PKCSReq
              pkiStatus {{id-attributes pkiStatus(3) "SUCCESS"}
              recipientNonce{{id-attribute recipientNonce(6)}<16 bytes>}
              senderNonce {{id-attribute senderNonce (5) 0x<16 bytes>}
          }
          digestEncryptionAlgorithm {pkcs-1 1}
          encryptedDigest "encrypted digest of authenticatedAttributes
                           using CA private key"
      }
  }
```

NOTE:The PKCS#7 EncryptedContent is specified as an octet string, but SCEP entities must also accept a sequence of octet strings as a valid alternate encoding.

This alternate encoding must be accepted wherever PKCS #7 Enveloped Data is specified in this document.

5.5 Get Certificate Authority Certificate

Before any transaction begins, end entities have to get the CA (and possibly RA) certificate(s) first. Since no public keys have been exchanged, the message can not be encrypted and the response must be authenticated by out-of-band means. These certs are obtained by means of an HTTP GET message. To get the CA certificate, the end entity does a "HTTP GET" and receives a plain X.509 certificate in response. In the request, the URL identifies a CGI script on the server and passes the CA issuer identifier as the parameter to the CGI script. Once the CA certificate is received by the end entity, a fingerprint is generated using MD5 hash algorithm on the whole CA certificate. This fingerprint is verified by some positive out-of-band means, such as a phone call.

5.5.1 GetCACert HTTP Message Format

"GET" CGI-PATH CGI-PROG "?operation=GetCACert" "&message=" CA-IDENT

where:

CGI-PATH defines the actual CGI path to invoke the CGI program which parses the request.

CGI-PROG is set to be the string "pkiclient.exe" and this is expected to be the program that the CA will use to handle the SCEP transactions.

CA-IDENT is any string which is understood by the CA.
For example, it could be a domain name like ietf.org.
If a certificate authority has multiple root certificates this field can be used to distinguish which is required.
Otherwise it may be ignored.

5.5.2 Response

The response for GetCACert is different between the case where the CA directly communicated with the end entity during the enrollment, and the case where a RA exists and the end entity communicates with the RA during the enrollment.

5.5.2.1 CA Certificate Only Response

A binary X.509 CA certificate is send back as a MIME object with a Content-Type of application/x-x509-ca-cert.

5.5.2.2 CA and RA Certificates Response

When an RA exists, both CA and RA certificates must be sent back in the response to the GetCACert request. The RA certificate(s) must be signed by the CA. A certificates-only PKCS#7 SignedData is used to carry the certificates to the end entity, with a Content-Type of application/x-x509-ca-ra-cert.

|5.6 Get Certificate Authority Certificate Chain
|
|In order to support Certificate Authority hierarchies, it is necessary
|to have a way to get the entire certificate chain. The following message
|has been added to SCEP for this purpose.
|
|5.6.1 GetCACertChain HTTP Message Format
|
| "GET" CGI-SCRIPT "?" "operation=GetCACertChain" "&" "message" CA-IDENT
| where CGI-SCRIPT and CA-IDENT are as described for GetCACert.
|
|5.6.2 Response
|
|The response for GetCACertChain is a certificates-only PKCS#7 SignedData
|to carry the certificates to the end entity, with a Content-Type of
|application/x-x509-ca-ra-cert-chain.
|
|5.6.3 Backwards Compatability
|
|Versions of SCEP prior to revision 3 do not support GetCACertChain.
|Certificate Authorities written to these prior versions will not be
|able to process the message and may return an HTML error.
|
|To avoid this, clients should send the GetCACert message first. If the
|returned certificate is self-signed or is signed by a Certificate
|Authority that is trusted by the client, then it is not necessary to
|send the GetCACertChain message and it should not be sent.
|
|If a Certificate Authority is configured with a certificate that is
|not either self-signed or has a self-signed issuer, then it should
|support this message. In other words, it should be supported if the
|CA hierarchy is more than two-deep.
|
|An old CA in a two-deep hierarchy might still get this message from
|a client if the client did not trust either that CA or its issuer.
|In that event, the certificate cannot be trusted anyway. In any case
|the CA must not crash or hang upon the receipt of the message and the
|client must be able to handle whatever error is returned by the CA,
|including an HTML error or an ungraceful disconnect.

The following is the ASN.1 definition of Cert-Only PKCS#7:

```
  certOnly SignedData ::= {
      version 1
      digestAlgorithm {iso(1) member-body(2) US(840) rsadsi(113549)
                       digestAlgorithm(2) 5}

contentInfo {
          contentType {pkcs-7 1} -- data content identifier
          content  -- NULL
      }
      certificates   -- the RA and CA certificates.
  }

  CARACerts PKIMessage ::= {  -- special pki message sent in the clear
      contentType {pkcs-7 2}
      content certOnly
  }
```

6.0 Security Considerations

This entire document is about security. Common security considerations such as keeping private keys truly private and using adequate lengths for symmetric and asymmetric keys must be followed in order to maintain the security of this protocol.

7.0 Intellectual Property

This protcol includes the optional use of Certificate Revocation List Distribution Point (CRLDP) technology, which is a patented technology of Entrust Technologies, Inc. (Method for Efficient Management of Certificate Revocation Lists and Update Information (U.S. Patent 5,699,431)). Please contact Entrust Technologies, Inc. (www.entrust.com) for more information on licensing CRLDP technology.

8.0 References

[PKCS7] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, March 1998.

[PKCS10] Kaliski, B., "PKCS #10: Certification Request Syntax Version 1.5", RFC 2314, March 1998.

[RFC2459] Housley, R., ec. al., "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 2459, January 1999.

Appendix A: Cisco End Entity Subject Name Definition

The ip address and the FQDN of a SCEP client should be included in the V3 extension subjectAltName. When the subjectAltName extension attribute is present, both the subjectAltName fields and the subjectName field could have the IP address and the FQDN information.

When the X.500 directory is used by the CA to define the name space, the subject name defined above become a RDN which is part of DN binded to the end entity's public key in the certificate.

A sample of DN assigned by Entrust CA is given below (assume the same ciscoRouterAlice is used as the end entity defined subject name):

```
OU = InteropTesting, O = Entrust Technologies, C = CA
RDN = {"alice.cisco.com", "172.21.114.67", "22334455"}
```

Appendix B: IPSEC Client Enrollment Certificate Request

The following is the certificate enrollment request (PKCS#10) as created by Cisco VPN Client:

```
-----END NEW CERTIFICATE REQUEST-----
   0 30  439: SEQUENCE {
   4 30  288:   SEQUENCE {
   8 02    1:     INTEGER 0
  11 30   57:     SEQUENCE {
  13 31   55:       SET {
  15 30   53:         SEQUENCE {
  17 06    3:           OBJECT IDENTIFIER commonName (2 5 4 3)
  22 13   46:           PrintableString
           :             'For Xiaoyi, IPSEC attrs in alternate name
           :              extn'
           :           }
           :         }
           :       }
  70 30  158:     SEQUENCE {
  73 30   13:       SEQUENCE {
  75 06    9:         OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1
           :                                           1 1)
  86 05    0:         NULL
           :         }
  88 03  140:       BIT STRING 0 unused bits
           :         30 81 88 02 81 80 73 DB 1D D5 65 AA EF C7 D4 8E
           :         AA 6E EB 46 AC 91 2A 0F 50 51 17 AD 50 A2 2A F2
           :         CE BE F1 E4 22 8C D7 61 A1 6C 87 61 62 92 CB A6
           :         80 EA B4 0F 09 9D 18 5F 39 A3 02 0E DB 38 4C E4
           :         8A 63 2E 72 8B DC BE 9E ED 6C 1A 47 DE 13 1B 0F
           :         83 29 4D 3E 08 86 FF 08 2B 43 09 EF 67 A7 6B EA
           :         77 62 30 35 4D A9 0F 0F DF CC 44 F5 4D 2C 2E 19
           :         E8 63 94 AC 84 A4 D0 01 E1 E3 97 16 CD 86 64 18
           :                 [ Another 11 bytes skipped ]
           :       }
 231 A0   63:     [0] {
 233 30   61:       SEQUENCE {
 235 06    9:         OBJECT IDENTIFIER extensionReq (1 2 840 113549 1 9
           :                                           14)
 246 31   48:         SET {
 248 30   46:           SEQUENCE {
 250 30   44:             SEQUENCE {
 252 06    3:               OBJECT IDENTIFIER subjectAltName (2 5 29 17)
 257 04   37:               OCTET STRING
                             30 23 87 04 01 02 03 04 81 0D 65 6D 61 69
```

```
                            6C 40 69 72 65 2E 63 6F 6D 82 0C 66 71 64
                            6E 2E 69 72 65 2E 63 6F 6D
            :               }
            :             }
            :           }
            :         }
            :       }
            :     }

 296 30   13:   SEQUENCE {
 298 06    9:     OBJECT IDENTIFIER md5withRSAEncryption (1 2 840 113549
                                                          1 1 4)
 309 05    0:     NULL
            :     }
 311 03  129:   BIT STRING 0 unused bits
            :     19 60 55 45 7F 72 FD 4E E5 3F D2 66 B0 77 13 9A
            :     87 86 75 6A E1 36 C6 B6 21 71 68 BD 96 F0 B4 60
            :     95 8F 12 F1 65 33 16 FD 46 8A 63 19 90 40 B4 B7
            :     2C B5 AC 63 17 50 28 F0 CD A4 F0 00 4E D2 DE 6D
            :     C3 4F F5 CB 03 4D C8 D8 31 5A 7C 01 47 D2 2B 91
            :     B5 48 55 C8 A7 0B DD 45 D3 4A 8D 94 04 3A 6C B0
            :     A7 1D 64 74 AB 8A F7 FF 82 C7 22 0A 2A 95 FB 24
            :     88 AA B6 27 83 C1 EC 5E A0 BA 0C BA 2E 6D 50 C7
            :   }
```

Appendix C: Private OID Definitions

The OIDs used in defining pkiStatus are VeriSign self-maintained OIDs. Please note, work is in progress to replace the VeriSign owned object identifiers with the standard object identifiers. Once the standarlization is completed, this documentation will be updated.

```
id-VeriSign   OBJECT_IDENTIFIER ::= {2 16 US(840) 1 VeriSign(113733)}
id-pki        OBJECT_IDENTIFIER ::= {id-VeriSign pki(1)}
id-attributes OBJECT_IDENTIFIER ::= {id-pki attributes(9)}
id-messageType  OBJECT_IDENTIFIER ::= {id-attributes messageType(2)}
id-pkiStatus    OBJECT_IDENTIFIER ::= {id-attributes pkiStatus(3)}
id-failInfo     OBJECT_IDENTIFIER ::= {id-attributes failInfo(4)}
id-senderNonce  OBJECT_IDENTIFIER ::= {id-attributes senderNonce(5)}
id-recipientNonce OBJECT_IDENTIFIER ::= {id-attributes recipientNonce(6)}
id-transId        OBJECT_IDENTIFIER ::= {id-attributes transId(7)}
id-extensionReq   OBJECT_IDENTIFIER ::= {id-attributes extensionReq(8)}
```

Appendix D: CRL Query by means of LDAP

In order to retrieve the CRL by means of LDAP, the client needs to know where in the directory it is stored. The certificate must contain a CRL Distribution Point extension encoded as a DN or as an LDAP URI.

For example, the certificate issued by Entrust VPN contains the following DN as the CRL distribution point:

CN = CRL1, O = cisco, C = US.

The asn.1 encoding of this distribution point is:

```
30 2C 31 0B 30 09 06 03 55 04 06 13 02 55 53 31 0E 30 0C 06
03 55 04 0A 13 05 63 69 73 63 6F 31 0D 30 0B 06 03 55 04 03
13 04 43 52 4C 31
```

The ldap form would be:

ldap://servername/CN=CRL1,O=cisco,C=US

Appendix E: SCEP State Transitions

SCEP state transitions are based on transaction identifier. The design goal is to ensure the synchronization between the CA and the end entity under various error situations.

An identity is defined by the combination of FQDN, the IP address and the client serial number. FQDN is the required name attribute. It is important to notice that, a client named as Alice.cisco.com is different from the client named as Alice.cisco.com plus IPAddress 171.69.1.129.

Each enrollment transaction is uniquely associated with a transaction identifier. Because the enrollment transaction could be interrupted by various errors, including network connection errors or client reboot, the SCEP client generates a transaction identifier by calculating MD5 hash on the public key value for which the enrollment is requested. This retains the same transaction identifier throughout the enrollment transaction, even if the client has rebooted or timed out, and issues a new enrollment request for the same key pair. It also provides the way for the CA to uniquely identify a transaction in its database. At the end entity side, it generates a transaction identifier which is included in PKCSReq. If the CA returns a response of PENDING, the end entity will poll by periodically sending out GetCertInitial with the same

transaction identifier until either a response other than PENDING is obtained, or the configured maximum time has elapsed.

If the client times out or the client reboots, the client administrator will start another enrollment transaction with the same key pair. The second enrollment will have the transaction idenifier. At the server side, instead of accepting the PKCSReq as a new enrollment request, it should respond as if another GetCertInitial message had been sent with that transaction ID. In another word, the second PKCSReq should be taken as a resynchronization message to allow the enrollment resume as the same transaction.

It is important to keep the transaction id unique since CEP requires the same policy and same identity be applied to the same subject name and

key pair binding. In the current implementation, an SCEP client can only assume one identity. At any time, only one key pair, with a given key usage, can be associated with the same identity.

The following gives several example of client to CA transactions.

Client actions are indicated in the left column, CA actions are indicated in the right column. A blank action signifies that no message was received. Note that these examples assume that the CA enforces the certificate-name uniqueness property defined in Section 2.1.1.1.

The first transaction, for example, would read like this:
"Client Sends PKCSReq message with transaction ID 1 to the CA. The CA signs the certificate and constructs a CertRep Message containing the signed certificate with a transaction ID 1. The client receives the message and installs the cert locally."

Successful Enrollment Case: no manual authentication

```
PKCSReq (1)             ----------> CA Signs Cert
Client Installs Cert    <---------- CertRep (1) SIGNED CERT
```

Successful Enrollment Case: manual authentication required

```
PKCSReq (10)            ----------> Cert Request goes into Queue
Client Polls            <---------- CertRep (10) PENDING
GetCertInitial (10)     ----------> Still pending
Client Polls            <---------- CertRep (10) PENDING
GetCertInitial (10)     ----------> Still pending
Client Polls            <---------- CertRep (10) PENDING
GetCertInitial (10)     ----------> Still pending
Client Polls            <---------- CertRep (10) PENDING
GetCertInitial (10)     ----------> Cert has been signed
Client Installs Cert    <---------- CertRep (10) SIGNED CERT
```

Resync Case - CA Receive and Signs PKCSReq, Client Did not receive CertRep:

```
PKCSReq (3)           ----------> Cert Request goes into queue
                      <---------- CertRep (3) PENDING
GetCertInitial (3)    ---------->
                      <---------- CertRep (3) PENDING
GetCertInitial (3)    ---------->
                      <---------- CA signed Cert and send back
                                  CertRep(3)
(Time Out)
PKCSReq (3)           ----------> Cert already signed, send back to
                                  client
```

```
Client Installs Cert  <---------- CertRep (3) SIGNED CERT
```

Case when NVRAM is lost and client has to generate a new key pair, there is no change of name information:

```
PKCSReq (4)           ----------> CA Signs Cert
Client Installs Cert  <---------- CertRep (4) SIGNED CERT
(Client looses Cert)
PKCSReq (5)           ----------> There is already a valid cert with
                                  this DN.
Client Admin Revokes  <---------- CertRep (5) OVERLAPPING CERT ERROR
PKCSReq (5)           ----------> CA Signs Cert
Client Installs Cert  <---------- CertRep (5) SIGNED CERT
```

Case when client admin resync the enrollment using a different PKCS#10:

```
PKCSReq (6)             ----------> CA Signs Cert
                        <---------- CertRep (6) SIGNED CERT
(Client timeout and admin starts another enrollment with a different
 PKCS#10, but the same transaction id)
PKCSReq (6)  with different PKCS#10
                        ----------> There is already a valid cert with
                                    this entity (by checking FQDN).
                        <---------- CertRep (6) INVALID PKCS#10 CERT
                                     ERROR
Client admin either revokes the existing cert
or corrects the error by enrolling with
the same PKCS#10 as the first PKCSReq(6)
PKCSReq (6)             ----------> CA find the existing Cert
Client Installs Cert    <---------- CertRep (6) SIGNED CERT
```

Resync case when server is slow in response:

```
PKCSReq (13)    ---------->  Cert Request goes into Queue
                <----------  CertRep (13) PENDING
GetCertInitial  ----------> Still pending
                <----------  CertRep (13) PENDING
GetCertInitial  ---------->  Still pending
                <----------  CertRep (13) PENDING
GetCertInitial  ---------->  Still pending
                <----------  CertRep (13) PENDING
GetCertInitial  ---------->  Still pending
(TimeOut)       <----------  CertRep (13) PENDING
* Case 1
PKCSReq (13)          ---------->  Still pending
Client polls          <----------  CertRep (13) PENDING
CertCertInitial       ---------->  Cert has been signed
Client Installs Cert  <----------  CertRep (13) SIGNED CERT
* Case 2
PKCSReq (13)          ----------> Cert has been signed
```

```
Client Installs Cert  <---------- CertRep (13) SIGNED CERT
```

Appendix F. Author Contact Information

Xiaoyi Liu
Cisco
170 West Tasman Drive
San Jose, CA 94134
xliu@cisco.com

Cheryl Madson
Cisco
170 West Tasman Drive
San Jose, CA 94134
cmadson@cisco.com

David McGrew
Cisco
170 West Tasman Drive
San Jose, CA 94134
mcgrew@cisco.com

Andrew Nourse
Cisco
101 Cooper Street
Santa Cruz, CA 95060
nourse@cisco.com

Appendix G. Copyright Section

Copyright (C) The Internet Society (2000). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implmentation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

[End of draft-nourse-scep-04.txt]