Stream: Internet Engineering Task Force (IETF)
RFC: 8894
Category: Informational
Published: September 2020
ISSN: 2070-1721
Author: P. Gutmann
University of Auckland

RFC 8894
Simple Certificate Enrolment Protocol

Abstract

This document specifies the Simple Certificate Enrolment Protocol (SCEP), a PKI protocol that leverages existing technology by using Cryptographic Message Syntax (CMS, formerly known as PKCS #7) and PKCS #10 over HTTP. SCEP is the evolution of the enrolment protocol sponsored by Cisco Systems, which enjoys wide support in both client and server implementations, as well as being relied upon by numerous other industry standards that work with certificates.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction
  1.1. Conventions Used in This Document
2. SCEP Overview
  2.1. SCEP Entities
    2.1.1. Client
    2.1.2. Certificate Authority
  2.2. CA Certificate Distribution
  2.3. Client Authentication
  2.4. Enrolment Authorisation
  2.5. Certificate Enrolment/Renewal
    2.5.1. Client State Transitions
  2.6. Certificate Access
  2.7. CRL Access
  2.8. Certificate Revocation
  2.9. Mandatory-to-Implement Functionality
3. SCEP Secure Message Objects
  3.1. SCEP Message Object Processing
  3.2. SCEP pkiMessage
    3.2.1. Signed Transaction Attributes
      3.2.1.1. transactionID
      3.2.1.2. messageType
      3.2.1.3. pkiStatus

3.2.1.4. failInfo and failInfoText

3.2.1.5. senderNonce and recipientNonce

3.2.2. SCEP pkcsPKIEnvelope

3.3. SCEP pkiMessage types

3.3.1. PKCSReq/RenewalReq

3.3.2. CertRep

3.3.2.1. CertRep SUCCESS

3.3.2.2. CertRep FAILURE

3.3.2.3. CertRep PENDING

3.3.3. CertPoll (GetCertInitial)

3.3.4. GetCert and GetCRL

3.4. Degenerate certificates-only CMS SignedData

3.5. CA Capabilities

3.5.1. GetCACaps HTTP Message Format

3.5.2. CA Capabilities Response Format

4. SCEP Transactions

4.1. HTTP POST and GET Message Formats

4.2. Get CA Certificate

4.2.1. Get CA Certificate Response Message Format

4.2.1.1. CA Certificate Response Message Format

4.2.1.2. CA Certificate Chain Response Message Format

4.3. Certificate Enrolment/Renewal

4.3.1. Certificate Enrolment/Renewal Response Message

4.4. Poll for Client Initial Certificate

4.4.1. Polling Response Message Format

4.5. Certificate Access

4.5.1. Certificate Access Response Message Format

4.6. CRL Access

4.6.1. CRL Access Response Message Format

- 4.7. Get Next Certificate Authority Certificate
  - 4.7.1. Get Next CA Response Message Format
- 5. SCEP Transaction Examples
  - 5.1. Successful Transactions
  - 5.2. Transactions with Errors
- 6. IANA Considerations
  - 6.1. Registration of the application/x-x509-ca-cert Media Type
  - 6.2. Registration of the application/x-x509-ca-ra-cert Media Type
  - 6.3. Registration of the application/x-x509-next-ca-cert Media Type
  - 6.4. Registration of the application/x-pki-message Media Type
- 7. Security Considerations
  - 7.1. General Security
  - 7.2. Use of the CA Private Key
  - 7.3. ChallengePassword Shared Secret Value
  - 7.4. Lack of Certificate Issue Confirmation
  - 7.5. GetCACaps Issues
  - 7.6. Lack of PoP in Renewal Requests
  - 7.7. Traffic Monitoring
  - 7.8. Unnecessary Cryptography
  - 7.9. Use of SHA-1
  - 7.10. Use of HTTP
- 8. References
  - 8.1. Normative References
  - 8.2. Informative References
- Appendix A. Background Notes
- Acknowledgements
- Author's Address

1. Introduction

X.509 certificates serve as the basis for several standardised security protocols such as TLS [RFC8446], S/MIME [RFC8551], and IKE/IPsec [RFC7296]. When an X.509 certificate is issued, there typically is a need for a certificate management protocol to enable a PKI client to request or renew a certificate from a Certificate Authority (CA). This specification defines a protocol, the Simple Certificate Enrolment Protocol (SCEP), for certificate management and certificate and CRL queries.

The SCEP protocol supports the following general operations:

- CA public key distribution
- Certificate enrolment and issue
- Certificate renewal
- Certificate query
- CRL query

SCEP makes extensive use of CMS [RFC5652] and PKCS #10 [RFC2986].

1.1. Conventions Used in This Document

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234] for defining formal syntax of commands. Non-terminals not defined in [RFC5234] are defined in Section 4.1.

2. SCEP Overview

This section provides an overview of the functionality of SCEP.

2.1. SCEP Entities

The entity types defined in SCEP are a client requesting a certificate and a Certificate Authority (CA) that issues the certificate. These are described in the following sections.

2.1.1. Client

A client **MUST** have the following information locally configured:

1. The CA's fully qualified domain name or IP address.
2. Any identification and/or authorisation information required by the CA before a certificate will be issued, as described in Section 3.3.1.

3. The identifying information that is used for authentication of the CA in Section 4.2.1, typically a certificate fingerprint.

2.1.2. Certificate Authority

A SCEP CA is the entity that signs client certificates. A CA may enforce policies and apply them to certificate requests, and it may reject a request for any reason.

Since the client is expected to perform signature verification and optionally encryption using the CA certificate, the keyUsage extension in the CA certificate **MUST** indicate that it is valid for digitalSignature and keyEncipherment (if the key is to be used for en/decryption) alongside the usual CA usages of keyCertSign and/or cRLSign.

2.2. CA Certificate Distribution

If the CA certificate(s) have not previously been acquired by the client through some other means, the client **MUST** retrieve them before any PKI operation (Section 3) can be started. Since no public key has yet been exchanged between the client and the CA, the messages cannot be secured using CMS, and the CA certificate request and response data is instead transferred in the clear.

If an intermediate CA is in use, a certificates-only CMS SignedData message with a certificate chain consisting of all CA certificates is returned. Otherwise, the CA certificate itself is returned.

The CA certificate **MAY** be provided out of band to the client. Alternatively, the CA certificate fingerprint **MAY** be used to authenticate a CA certificate distributed by the GetCACert response (Section 4.2) or via HTTP certificate-store access [RFC4387]. The fingerprint is created by calculating a SHA-256 hash over the whole CA certificate. (For legacy reasons, a SHA-1 hash may be used by some implementations.)

After the client gets the CA certificate, it **SHOULD** authenticate it in some manner unless this is deemed unnecessary, for example, because the device is being provisioned inside a trusted environment. For example, the client could compare the certificate's fingerprint with locally configured, out-of-band distributed, identifying information, or by some equivalent means such as a direct comparison with a locally stored copy of the certificate.

Intermediate CA certificates, if any, are signed by a higher-level CA, so there is no need to authenticate them against the out-of-band data. Since intermediate CA certificates are rolled over more frequently than long-lived top-level CA certificates, clients **MUST** verify intermediate-level CA certificates before use during protocol exchanges in case the intermediate CA certificate has expired or otherwise been invalidated.

When a CA certificate expires, certificates that have been signed by it may no longer be regarded as valid. CA key rollover provides a mechanism by which the CA can distribute a new CA certificate that will be valid in the future once the current certificate has expired. This is done via the GetNextCACert message (Section 4.7).

2.3. Client 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. Communications between the client and the CA are secured using SCEP Secure Message Objects as explained in Section 3, which specifies how CMS is used to encrypt and sign the data. In order to perform the signing operation, the client uses an appropriate local certificate:

1. If the client does not have an appropriate existing certificate, then a locally generated self-signed certificate **MUST** be used. The keyUsage extension in the certificate **MUST** indicate that it is valid for digitalSignature and keyEncipherment (if available). The self-signed certificate **SHOULD** use the same subject name and key as in the PKCS #10 request. In this case, the messageType is PKCSReq (see Section 3.2.1.2).
2. If the client already has a certificate issued by the SCEP CA, and the CA supports renewal (see Section 2.5), that certificate **SHOULD** be used. In this case, the messageType is RenewalReq (see Section 3.2.1.2).
3. Alternatively, if the client has no certificate issued by the SCEP CA but has credentials from an alternate CA, then the certificate issued by the alternate CA **MAY** be used in a renewal request as described above. The SCEP CA's policy will determine whether the request can be accepted or not.

Note that although the above text describes several different types of operations, for historical reasons, most implementations always apply the first one, even if an existing certificate already exists. For this reason, support for the first case is mandatory while support for the latter ones are optional (see Section 2.9).

During the certificate-enrolment process, the client **MUST** use the selected certificate's key when signing the CMS envelope (see Section 3). This certificate will be either the self-signed one matching the PKCS #10 request or the CA-issued one used to authorise a renewal, and it **MUST** be included in the signedData certificates field (possibly as part of a full certificate chain). If the key being certified allows encryption, then the CA's CertResp will use the same certificate's public key when encrypting the response.

Note that, in the case of renewal operations, this means that the request will be signed and authenticated with the key in the previously issued certificate rather than the key in the PKCS #10 request, and the response may similarly be returned encrypted with the key in the previously issued certificate. This has security implications; see Section 7.6.

2.4. Enrolment Authorisation

PKCS #10 [RFC2986] specifies a PKCS #9 [RFC2985] challengePassword attribute to be sent as part of the enrolment request. When utilising the challengePassword, the CA distributes a shared secret to the client, which will be used to authenticate the request from the client. It is

RECOMMENDED that the challengePassword be a one-time authenticator value to limit the ability of an attacker who can capture the authenticator from the client or CA and reuse it to request further certificates.

Inclusion of the challengePassword by the SCEP client is **RECOMMENDED**; however, its omission allows for unauthenticated authorisation of enrolment requests (which may, however, require manual approval of each certificate issue if other security measures to control issue aren't in place; see below). Inclusion is **OPTIONAL** for renewal requests that are authenticated by being signed with an existing certificate. The CMS envelope protects the privacy of the challengePassword.

A client that is performing certificate renewal as per Section 2.5 **SHOULD** omit the challengePassword but **MAY** send the originally distributed shared secret in the challengePassword attribute. The SCEP CA **MAY** authenticate the request using the challengePassword in addition to the previously issued certificate that signs the request. The SCEP CA **MUST NOT** attempt to authenticate a client based on a self-signed certificate unless it has been verified through out-of-band means such as a certificate fingerprint.

To perform the authorisation in manual mode, the client's request is placed in the PENDING state until the CA operator authorises or rejects it. Manual authorisation is used when the client has only a self-signed certificate that hasn't been previously authenticated by the CA and/or a challengePassword is not available. The SCEP CA **MAY** either reject unauthorised requests or mark them for manual authorisation according to CA policy.

2.5. Certificate Enrolment/Renewal

A client starts an enrolment transaction (Section 3.3.1) by creating a certificate request using PKCS #10 and sends the request to the CA enveloped using CMS (Section 3).

If the CA supports certificate renewal and the CA policy permits, then a new certificate with new validity dates can be issued, even though the old one is still valid. To renew an existing certificate, the client uses the RenewalReq message (see Section 3.3) and signs it with the existing client certificate. The client **SHOULD** use a new keypair when requesting a new certificate but **MAY** request a new certificate using the old keypair.

If the CA returns a CertRep message (Section 3.3.2) with status set to PENDING, the client enters into polling mode by periodically sending a CertPoll message (Section 3.3.3) to the CA until the CA operator completes the manual authentication (approving or denying the request). The frequency of the polling operation is a CA/client configuration issue and may range from seconds or minutes when the issue process is automatic but not instantaneous, through to hours or days if the certificate-issue operation requires manual approval.

If polling mode is being used, then the client will send a single PKCSReq/RenewalReq message (Section 3.3.1), followed by 0 or more CertPoll messages (Section 3.3.3). The CA will, in return, send 0 or more CertRep messages (Section 3.3.2) with status set to PENDING in response to CertPolls, followed by a single CertRep message (Section 3.3.2) with status set to either SUCCESS or FAILURE.

2.5.1. Client State Transitions

The client state transitions during the SCEP process are indicated in Figure 1.

```
                               CertPoll
                             +-----<----+
                             |          |
                             |          | CertRep(PENDING)
                             |          |
[CERT-NONEXISTENT] ------> [CERT-REQ-PENDING] --------> [CERT-ISSUED]
      ^             PKCSReq    |          CertRep(SUCCESS)
      |           RenewalReq   |
      |                        |
      +------------------------+
      CertRep(FAILURE) or
      Max-time/max-polls exceeded
```

Figure 1: State Transition Diagram

The certificate-issue process starts at state CERT-NONEXISTENT. Sending a PKCSReq/RenewalReq message changes the state to CERT-REQ-PENDING.

If the CA returns a CertRep message with pkiStatus set to SUCCESS, then the state changes to CERT-ISSUED.

If the CA returns a CertRep message with pkiStatus set to FAILURE or there is no response, then the state reverts back to CERT-NONEXISTENT.

If the CA returns a CertRep message with pkiStatus set to PENDING, then the client will keep polling by sending a CertPoll message until either a CertRep message with status set to SUCCESS or FAILURE is received, a timeout occurs, or the maximum number of polls has been exceeded.

Figure 2 shows a successful transaction in automatic mode

```
    CLIENT                                CA SERVER

PKCSReq: PKI cert. enrolment message
--------------------------------> CertRep: pkiStatus = SUCCESS
                                  Certificate attached
                                  <------------------------------
Receive issued certificate.
```

Figure 2: Automatic Mode

Figure 3 shows a successful transaction in manual mode:

```
   CLIENT                                CA SERVER

PKCSReq: PKI cert. enrolment message
--------------------------------> CertRep: pkiStatus = PENDING
                                  <------------------------------
CertPoll: Polling message
--------------------------------> CertRep: pkiStatus = PENDING
                                  <------------------------------
................ <Manual identity authentication> ...............

CertPoll: Polling message
--------------------------------> CertRep: pkiStatus = SUCCESS
                                  Certificate attached
                                  <------------------------------
Receive issued certificate.
```

Figure 3: Manual Mode

2.6. Certificate Access

A certificate query message is defined for clients to retrieve a copy of their own certificate from the CA. It allows clients that do not store their certificates locally to obtain a copy when needed. This functionality is not intended to provide a general-purpose certificate-access service, which may be achieved instead via HTTP certificate-store access [RFC4387] or Lightweight Directory Access Protocol (LDAP).

To retrieve a certificate from the CA, a client sends a request consisting of the certificate's issuer name and serial number. This assumes that the client has saved the issuer name and the serial number of the issued certificate from the previous enrolment transaction. The transaction to retrieve a certificate consists of one GetCert (Section 3.3.4) message and one CertRep (Section 3.3.2) message, as shown in Figure 4.

```
  CLIENT                                 CA SERVER

GetCert: PKI certificate query message
-------------------------------> CertRep: pkiStatus = SUCCESS
                                 Certificate attached
                                 <-----------------------------
Receive the certificate.
```

Figure 4: Retrieving a Certificate

2.7. CRL Access

SCEP clients **MAY** request a CRL via one of three methods:

1. If the CA supports the CRL Distribution Points (CRLDPs) extension [RFC5280] in issued certificates, then the CRL **MAY** be retrieved via the mechanism specified in the CRLDP.

2. If the CA supports HTTP certificate-store access [RFC4387], then the CRL **MAY** be retrieved via the AuthorityInfoAcces [RFC5280] location specified in the certificate.
3. Only if the CA does not support CRLDPs or HTTP access should a CRL query be composed by creating a GetCRL message consisting of the issuer name and serial number from the certificate whose revocation status is being queried.

The message is sent to the SCEP CA in the same way as the other SCEP requests. The transaction to retrieve a CRL consists of one GetCRL PKI message and one CertRep PKI message, which contains only the CRL (no certificates) in a degenerate certificates-only CMS SignedData message (Section 3.4), as shown in Figure 5.

```
      CLIENT                          CA SERVER

   GetCRL: PKI CRL query message
---------------------------------->
                                  CertRep: CRL attached
                                  <-----------------------------
Receive the CRL
```

Figure 5: Retrieving a CRL

2.8. Certificate Revocation

SCEP does not specify a method to request certificate revocation. In order to revoke a certificate, the client must contact the CA using a non-SCEP-defined mechanism.

2.9. Mandatory-to-Implement Functionality

At a minimum, all SCEP implementations compliant with this specification **MUST** support GetCACaps (Section 3.5.1), GetCACert (Section 4.2), PKCSReq (Section 3.3.1) (and its associated response messages), communication of binary data via HTTP POST (Section 4.1), and the AES128-CBC [AES] and SHA-256 [SHA2] algorithms to secure pkiMessages (Section 3.2).

For historical reasons, implementations **MAY** support communications of binary data via HTTP GET (Section 4.1), and the triple DES-CBC and SHA-1 algorithms to secure pkiMessages (Section 3.2). Implementations **MUST NOT** support the obsolete and/or insecure single DES and MD5 algorithms used in earlier versions of this specification, since the unsecured nature of GetCACaps means that an in-path attacker can trivially roll back the encryption used to these insecure algorithms; see Section 7.5.

3. SCEP Secure Message Objects

CMS is a general enveloping mechanism that enables both signed and encrypted transmission of arbitrary data. SCEP messages that require confidentiality use two layers of CMS, as shown using ASN.1-like pseudocode in Figure 6. By applying both enveloping and signing transformations, the SCEP message is protected both for the integrity of its end-to-end transaction information and the confidentiality of its information portion.

```
pkiMessage {
  contentType = signedData { pkcs-7 2 },
  content {
    digestAlgorithms,
    encapsulatedContentInfo {
      eContentType = data { pkcs-7 1 },
      eContent {           -- pkcsPKIEnvelope, optional
        contentType = envelopedData { pkcs-7 3 },
        content {
          recipientInfo,
          encryptedContentInfo {
            contentType = data { pkcs-7 1 },
            contentEncrAlgorithm,
            encryptedContent {
              messageData  -- Typically PKCS #10 request
              }
            }
          }
        }
      },
    certificates,          -- Optional
    crls,                  -- Optional
    signerInfo {
      signedAttrs {
        transactionID,
        messageType,
        pkiStatus,
        failInfo,          -- Optional
        senderNonce / recipientNonce,
        },
      signature
      }
    }
  }
```

Figure 6: CMS Layering

When a particular SCEP message carries data, this data is carried in the messageData. CertRep messages will lack any signed content and consist only of a pkcsPKIEnvelope (Section 3.2.2).

The remainder of this document will refer only to "messageData", but it is understood to always be encapsulated in the pkcsPKIEnvelope (Section 3.2.2). The format of the data in the messageData is defined by the messageType attribute (see Section 3.2) of the SignedData. If there is no messageData to be transmitted, the entire pkcsPKIEnvelope **MUST** be omitted.

Samples of SCEP messages are available through the JSCEP project [JSCEP] in the src/samples directory.

3.1. SCEP Message Object Processing

Creating a SCEP message consists of several stages. The content to be conveyed (in other words, the messageData) is first encrypted, and the encrypted content is then signed.

The form of encryption to be applied depends on the capabilities of the recipient's public key. If the key is encryption capable (for example, RSA), then the messageData is encrypted using the recipient's public key with the CMS KeyTransRecipientInfo mechanism. If the key is not encryption capable (for example, DSA or ECDSA), then the messageData is encrypted using the challengePassword with the CMS PasswordRecipientInfo mechanism.

Once the messageData has been encrypted, it is signed with the sender's public key. This completes the SCEP message, which is then sent to the recipient.

Note that some early implementations of this specification dealt with keys that were not encryption capable by omitting the encryption stage, based on the text in Section 3 that indicated that "the EnvelopedData is omitted". This alternative processing mechanism **SHOULD NOT** be used since it exposes in cleartext the challengePassword used to authorise the certificate issue.

3.2. SCEP pkiMessage

The basic building block of all secured SCEP messages is the SCEP pkiMessage. It consists of a CMS SignedData content type. The following restrictions apply:

- The eContentType in encapsulatedContentInfo **MUST** be data ({pkcs-7 1}).
- The signed content, if present (FAILURE and PENDING CertRep messages will lack any signed content), **MUST** be a pkcsPKIEnvelope (Section 3.2.2) and **MUST** match the messageType attribute.
- The SignerInfo **MUST** contain a set of authenticatedAttributes (Section 3.2.1).

3.2.1. Signed Transaction Attributes

At a minimum, all messages **MUST** contain the following authenticatedAttributes:

- A transactionID attribute (see Section 3.2.1.1).
- A messageType attribute (see Section 3.2.1.2).
- A fresh senderNonce attribute (see Section 3.2.1.5). However, note the comment about senderNonces and polling in Section 3.3.2
- Any attributes required by CMS.

If the message is a CertRep, it **MUST** also include the following authenticatedAttributes:

- A pkiStatus attribute (see Section 3.2.1.3).
- failInfo and optional failInfoText attributes (see Section 3.2.1.4) if pkiStatus = FAILURE.
- A recipientNonce attribute (see Section 3.2.1.5) copied from the senderNonce in the request that this is a response to.

The following transaction attributes are encoded as authenticated attributes and carried in the SignerInfo for this SignedData.

Attribute	Encoding	Comment
transactionID	PrintableString	Unique ID for this transaction as a text string
messageType	PrintableString	Decimal value as a numeric text string
pkiStatus	PrintableString	Decimal value as a numeric text string
failInfo	PrintableString	Decimal value as a numeric text string
failInfoText	UTF8String	Descriptive text for the failInfo value
senderNonce	OCTET STRING	Random nonce as a 16-byte binary data string
recipientNonce	OCTET STRING	Random nonce as a 16-byte binary data string

Table 1: SCEP Attributes

The OIDs used for these attributes are as follows:

Name	ASN.1 Definition
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-transactionID	OBJECT_IDENTIFIER ::= {id-attributes transactionID(7)}
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-scep	OBJECT IDENTIFIER ::= {id-pkix 24}
id-scep-failInfoText	OBJECT IDENTIFIER ::= {id-scep 1}

Table 2: SCEP Attribute OIDs

The attributes are detailed in the following sections.

3.2.1.1. transactionID

A PKI operation is a transaction consisting of the messages exchanged between a client and the CA. The transactionID is a text string provided by the client when starting a transaction. The client **MUST** use a unique string as the transaction identifier, encoded as a PrintableString, which **MUST** be used for all PKI messages exchanged for a given operation, such as a certificate issue.

Note that the transactionID must be unique, but not necessarily randomly generated. For example, it may be a value assigned by the CA to allow the client to be identified by their transactionID, using a value such as the client device's Extended Unique Identifier (EUI), Remote Terminal Unit (RTU) ID, or a similar unique identifier. This can be useful when the client doesn't have a preassigned Distinguished Name through which the CA can identify their request -- for example, when enrolling Supervisory Control and Data Acquisition (SCADA) devices.

3.2.1.2. messageType

The messageType attribute specifies the type of operation performed by the transaction. This attribute **MUST** be included in all PKI messages. The following message types are defined:

Value	Name	Description
0	Reserved	
3	CertRep	Response to certificate or CRL request.
17	RenewalReq	PKCS #10 certificate request authenticated with an existing certificate.
19	PKCSReq	PKCS #10 certificate request authenticated with a shared secret.
20	CertPoll	Certificate polling in manual enrolment.
21	GetCert	Retrieve a certificate.
22	GetCRL	Retrieve a CRL.

Table 3: SCEP Message Types

Message types not defined above **MUST** be treated as errors unless their use has been negotiated through GetCACaps (Section 3.5.1).

3.2.1.3. pkiStatus

All response messages **MUST** include transaction status information, which is defined as a pkiStatus attribute:

Value	Name	Description
0	SUCCESS	Request granted.

Value	Name	Description
2	FAILURE	Request rejected. In this case, the failInfo attribute, as defined in Section 3.2.1.4, **MUST** also be present.
3	PENDING	Request pending for manual approval.

Table 4: pkiStatus Attributes

PKI status values not defined above **MUST** be treated as errors unless their use has been negotiated through GetCACaps (Section 3.5.1).

3.2.1.4. failInfo and failInfoText

The failInfo attribute **MUST** contain one of the following failure reasons:

Value	Name	Description
0	badAlg	Unrecognised or unsupported algorithm.
1	badMessageCheck	Integrity check (meaning signature verification of the CMS message) failed.
2	badRequest	Transaction not permitted or supported.
3	badTime	The signingTime attribute from the CMS authenticatedAttributes was not sufficiently close to the system time. This condition may occur if the CA is concerned about replays of old messages.
4	badCertId	No certificate could be identified matching the provided criteria.

Table 5: failInfo Attributes

Failure reasons not defined above **MUST** be treated as errors unless their use has been negotiated through GetCACaps (Section 3.5.1).

The failInfoText is a free-form UTF-8 text string that provides further information in the case of pkiStatus = FAILURE. In particular, it may be used to provide details on why a certificate request was not granted that go beyond what's provided by the near-universal failInfo = badRequest status. Since this is a free-form text string intended for interpretation by humans, implementations **SHOULD NOT** assume that it has any type of machine-processable content.

3.2.1.5. senderNonce and recipientNonce

The senderNonce and recipientNonce attributes are each a 16-byte random number generated for each transaction. These are intended to prevent replay attacks.

When a sender sends a PKI message to a recipient, a fresh senderNonce **MUST** be included in the message. The recipient **MUST** copy the senderNonce into the recipientNonce of the reply as a proof of liveliness. The original sender **MUST** verify that the recipientNonce of the reply matches the senderNonce it sent in the request. If the nonce does not match, then the message **MUST** be rejected.

Note that since SCEP exchanges consist of a single request followed by a single response, the use of distinct sender and recipient nonces is redundant, since the client sends a nonce in its request and the CA responds with the same nonce in its reply. In effect, there's just a single nonce, identified as senderNonce in the client's request and recipientNonce in the CA's reply.

3.2.2. SCEP pkcsPKIEnvelope

The information portion of a SCEP message is carried inside an EnvelopedData content type, as defined in CMS, with the following restrictions:

- contentType in encryptedContentInfo **MUST** be data ({pkcs-7 1}).
- encryptedContent **MUST** be the SCEP message being transported (see Section 4) and **MUST** match the messageType authenticated Attribute in the pkiMessage.

3.3. SCEP pkiMessage types

All of the messages in this section are pkiMessages (Section 3.2), where the type of the message **MUST** be specified in the "messageType" authenticated Attribute. Each section defines a valid message type, the corresponding messageData formats, and mandatory authenticated attributes for that type.

3.3.1. PKCSReq/RenewalReq

The messageData for this type consists of a PKCS #10 Certificate Request. The certificate request **MUST** contain at least the following items:

- The subject Distinguished Name.
- The subject public key.
- For a PKCSReq, if authorisation based on a shared secret is being used, a challengePassword attribute.

In addition, the message must contain the authenticatedAttributes specified in Section 3.2.1.

3.3.2. CertRep

The messageData for this type consists of a degenerate certificates-only CMS SignedData message (Section 3.4). The exact content required for the reply depends on the type of request that this message is a response to. The request types are detailed in Sections 3.3.2.1 and 4. In addition, the message must contain the authenticatedAttributes specified in Section 3.2.1.

Earlier draft versions of this specification required that this message include a senderNonce alongside the recipientNonce, which was to be used to chain to subsequent polling operations. However, if a single message was lost during the potentially extended interval over which polling could take place (see Section 5 for an example of this), then if the implementation were to enforce this requirement, the overall transaction would fail, even though nothing had actually gone wrong. Because of this issue, implementations mostly ignored the requirement to either carry this nonce over to subsequent polling messages or verify its presence. More recent versions of the specification no longer require the chaining of nonces across polling operations.

3.3.2.1. CertRep SUCCESS

When the pkiStatus attribute is set to SUCCESS, the messageData for this message consists of a degenerate certificates-only CMS SignedData message (Section 3.4). The content of this degenerate certificates-only SignedData message depends on what the original request was, as outlined in Table 6.

Request-type	Reply-contents
PKCSReq	The reply **MUST** contain at least the issued certificate in the certificates field of the SignedData. The reply **MAY** contain additional certificates, but the issued certificate **MUST** be the leaf certificate.
RenewalReq	Same as PKCSReq
CertPoll	Same as PKCSReq
GetCert	The reply **MUST** contain at least the requested certificate in the certificates field of the SignedData. The reply **MAY** contain additional certificates, but the requested certificate **MUST** be the leaf certificate.
GetCRL	The reply **MUST** contain the CRL in the crls field of the SignedData.

Table 6: CertRep Response Types

3.3.2.2. CertRep FAILURE

When the pkiStatus attribute is set to FAILURE, the reply **MUST** also contain a failInfo (Section 3.2.1.4) attribute set to the appropriate error condition describing the failure. The reply **MAY** also contain a failInfoText attribute providing extended details on why the operation failed, typically to expand on the catchall failInfo = badRequest status. The pkcsPKIEnvelope (Section 3.2.2) **MUST** be omitted.

3.3.2.3. CertRep PENDING

When the pkiStatus attribute is set to PENDING, the pkcsPKIEnvelope (Section 3.2.2) **MUST** be omitted.

3.3.3. CertPoll (GetCertInitial)

This message is used for certificate polling. For unknown reasons, it was referred to as "GetCertInitial" in earlier draft versions of this specification. The messageData for this type consists of an IssuerAndSubject:

```
issuerAndSubject ::= SEQUENCE {
    issuer     Name,
    subject    Name
    }
```

The issuer is set to the subjectName of the CA (in other words, the intended issuerName of the certificate that's being requested). The subject is set to the subjectName used when requesting the certificate.

Note that both of these fields are redundant; the CA is identified by the recipientInfo in the pkcsPKIEnvelope (or in most cases, simply by the server that the message is being sent to), and the client/transaction being polled is identified by the transactionID. Both of these fields can be processed by the CA without going through the cryptographically expensive process of unwrapping and processing the issuerAndSubject. For this reason, implementations **SHOULD** assume that the polling operation will be controlled by the recipientInfo and transactionID rather than the contents of the messageData. In addition, the message must contain the authenticatedAttributes specified in Section 3.2.1.

3.3.4. GetCert and GetCRL

The messageData for these types consist of an IssuerAndSerialNumber, as defined in CMS, that uniquely identifies the certificate being requested, either the certificate itself for GetCert or its revocation status via a CRL for GetCRL. In addition, the message must contain the authenticatedAttributes specified in Section 3.2.1.

These message types, while included here for completeness, apply unnecessary cryptography and messaging overhead to the simple task of transferring a certificate or CRL (see Section 7.8). Implementations **SHOULD** prefer HTTP certificate-store access [RFC4387] or LDAP over the use of these messages.

3.4. Degenerate certificates-only CMS SignedData

CMS includes a degenerate case of the SignedData content type in which there are no signers. The use of such a degenerate case is to disseminate certificates and CRLs. For SCEP, the content field of the ContentInfo value of a degenerate certificates-only SignedData **MUST** be omitted. When carrying certificates, the certificates are included in the certificates field of the SignedData. When carrying a CRL, the CRL is included in the crls field of the SignedData.

3.5. CA Capabilities

In order to provide support for future enhancements to the protocol, CAs **MUST** implement the GetCACaps message to allow clients to query which functionality is available from the CA.

3.5.1. GetCACaps HTTP Message Format

This message requests capabilities from a CA, with the format as described in Section 4.1:

```
"GET" SP SCEPPATH "?operation=GetCACaps" SP HTTP-version CRLF
```

3.5.2. CA Capabilities Response Format

The response for a GetCACaps message is a list of CA capabilities, in plain text and in any order, separated by <CR><LF> or <LF> characters. This specification defines the following keywords (quotation marks are not sent):

Keyword	Description
AES	CA supports the AES128-CBC encryption algorithm.
DES3	CA supports the triple DES-CBC encryption algorithm.
GetNextCACert	CA supports the GetNextCACert message.
POSTPKIOperation	CA supports PKIOPeration messages sent via HTTP POST.
Renewal	CA supports the Renewal CA operation.
SHA-1	CA supports the SHA-1 hashing algorithm.
SHA-256	CA supports the SHA-256 hashing algorithm.
SHA-512	CA supports the SHA-512 hashing algorithm.
SCEPStandard	CA supports all mandatory-to-implement sections of the SCEP standard. This keyword implies "AES", "POSTPKIOperation", and "SHA-256", as well as the provisions of Section 2.9.

Table 7: GetCACaps Response Keywords

Table 7 lists all of the keywords that are defined in this specification. A CA **MAY** provide additional keywords advertising further capabilities and functionality. A client **MUST** be able to accept and ignore any unknown keywords that might be sent by a CA.

The CA **MUST** use the text case specified here, but clients **SHOULD** ignore the text case when processing this message. Clients **MUST** accept the standard HTTP-style text delimited by <CR><LF> as well as the text delimited by <LF> specified in an earlier draft version of this specification.

The client **SHOULD** use SHA-256 in preference to SHA-1 hashing and AES128-CBC in preference to triple DES-CBC if they are supported by the CA. Although the CMS format allows any form of AES and SHA-2 to be specified, in the interests of interoperability the de facto universal standards of AES128-CBC and SHA-256 **SHOULD** be used.

Announcing some of these capabilities individually is redundant, since they're required as mandatory-to-implement functionality (see Section 2.9) whose presence as a whole is signalled by the "SCEPStandard" capability. However, it may be useful to announce them in order to deal with older implementations that would otherwise default to obsolete, insecure algorithms and mechanisms.

If the CA supports none of the above capabilities, it **SHOULD** return an empty message. A CA **MAY** simply return an HTTP error. A client that receives an empty message or an HTTP error **SHOULD** interpret the response as if none of the capabilities listed are supported by the CA.

Note that at least one widely deployed server implementation supports several of the above operations but doesn't support the GetCACaps message to indicate that it supports them, and it will close the connection if sent a GetCACaps message. This means that the equivalent of GetCACaps must be performed through server fingerprinting, which can be done using the ID string "Microsoft-IIS". Newer versions of the same server, if sent a SCEP request using AES and SHA-2, will respond with an invalid response that can't be decrypted, requiring the use of 3DES and SHA-1 in order to obtain a response that can be processed, even if AES and/or SHA-2 are allegedly supported. In addition, the server will generate CA certificates that only have one, but not both, of the keyEncipherment and digitalSignature keyUsage flags set, requiring that the client ignore the keyUsage flags in order to use the certificates for SCEP.

The Content-type of the reply **SHOULD** be "text/plain". Clients **SHOULD** ignore the Content-type, as older implementations of SCEP may send various Content-types.

Example:

```
GET /cgi-bin/pkiclient.exe?operation=GetCACaps HTTP/1.1
```

might return:

```
AES
GetNextCACert
POSTPKIOperation
SCEPStandard
SHA-256
```

This means that the CA supports modern crypto algorithms, and the GetNextCACert message allows PKIOperation messages (PKCSReq/RenewalReq, GetCert, CertPoll, ...) to be sent using HTTP POST and is compliant with the final version of the SCEP standard.

4. SCEP Transactions

This section describes the SCEP Transactions and their HTTP [RFC7230] transport mechanism.

Note that SCEP doesn't follow best current practices on usage of HTTP. In particular, it recommends ignoring some media types and hard-codes specific URI paths. Guidance on the appropriate application of HTTP in these circumstances may be found in [HTTP].

4.1. HTTP POST and GET Message Formats

SCEP uses the HTTP POST and GET methods [RFC7230] to exchange information with the CA. The following defines the ABNF syntax of HTTP POST and GET methods sent from a client to a CA:

```
POSTREQUEST = "POST" SP SCEPPATH "?operation=" OPERATION
              SP HTTP-version CRLF

GETREQUEST = "GET" SP SCEPPATH "?operation=" OPERATION
             "&message=" MESSAGE SP HTTP-version CRLF
```

where:

- SCEPPATH is the HTTP URL path for accessing the CA. Clients **SHOULD** set SCEPPATH to the fixed string "/cgi-bin/pkiclient.exe" unless directed to do otherwise by the CA.
- OPERATION depends on the SCEP transaction and is defined in the following sections.
- HTTP-version is the HTTP version string, which is "HTTP/1.1" for [RFC7230].
- SP and CRLF are space and carriage return/linefeed, as defined in [RFC5234].

The CA will typically ignore SCEPPATH, since it's unlikely to be issuing certificates via a web server. Clients **SHOULD** set SCEPPATH to the fixed string "/cgi-bin/pkiclient.exe" unless directed to do otherwise by the CA. The CA **SHOULD** ignore the SCEPPATH unless its precise format is critical to the CA's operation.

Early SCEP drafts performed all communications via GET messages, including non-idempotent ones that should have been sent via POST messages; see [HTTP] for details. This has caused problems because of the way that the (supposedly) idempotent GET interacts with caches and proxies, and because the extremely large GET requests created by encoding CMS messages may be truncated in transit. These issues are typically not visible when testing on a LAN, but crop up during deployment over WANs. If the remote CA supports POST, the CMS-encoded SCEP messages **MUST** be sent via HTTP POST instead of HTTP GET. This applies to any SCEP message except GetCACert, GetNextCACert, and GetCACaps and avoids the need for base64 and URL encoding that's required for GET messaging. The client can verify that the CA supports SCEP messages via POST by looking for the "SCEPStandard" or "POSTPKIOperation" capability (see Section 3.5.2).

If a client or CA uses HTTP GET and encounters HTTP-related problems such as messages being truncated, seeing errors such as HTTP 414 ("Request-URI too long"), or simply having the message not sent/received at all when standard requests to the server (for example, via a web browser) work, then this is a symptom of the problematic use of HTTP GET. The solution to this problem is to update the implementation to use HTTP POST instead. In addition, when using GET, it's recommended to test the implementation from as many different network locations as possible to determine whether the use of GET will cause problems with communications.

When using GET messages to communicate binary data, base64 encoding as specified in Section 4 of [RFC4648] **MUST** be used. The base64-encoded data is distinct from "base64url" and may contain URI reserved characters; thus, it **MUST** be escaped as specified in [RFC3986] in addition to being base64 encoded. Finally, the encoded data is inserted into the MESSAGE portion of the HTTP GET request.

4.2. Get CA Certificate

To get the CA certificate(s), the client sends a GetCACert message to the CA. The OPERATION **MUST** be set to "GetCACert". There is no request data associated with this message.

4.2.1. Get CA Certificate Response Message Format

The response for GetCACert is different between the case where the CA directly communicates with the client during the enrolment and the case where an intermediate CA exists and the client communicates with this CA during the enrolment.

4.2.1.1. CA Certificate Response Message Format

If the CA does not have any intermediate CA certificates, the response consists of a single X.509 CA certificate. The response will have a Content-Type of "application/x-x509-ca-cert".

```
"Content-Type: application/x-x509-ca-cert"

<binary X.509>
```

4.2.1.2. CA Certificate Chain Response Message Format

If the CA has intermediate CA certificates, the response consists of a degenerate certificates-only CMS SignedData message (Section 3.4) containing the certificates, with the intermediate CA certificate(s) as the leaf certificate(s). The response will have a Content-Type of "application/x-x509-ca-ra-cert". Note that this designation is used for historical reasons due to its use in older versions of this specification -- no special meaning should be attached to the label.

```
"Content-Type: application/x-x509-ca-ra-cert"

<binary CMS>
```

4.3. Certificate Enrolment/Renewal

A PKCSReq/RenewalReq (Section 3.3.1) message is used to perform a certificate enrolment or renewal transaction. The OPERATION **MUST** be set to "PKIOperation". Note that when used with HTTP POST, the only OPERATION possible is "PKIOperation", so many CAs don't check this value or even notice its absence. When implemented using HTTP POST, the message is sent with a Content-Type of "application/x-pki-message" and might look as follows:

```
POST /cgi-bin/pkiclient.exe?operation=PKIOperation HTTP/1.1
Content-Length: <length of data>
Content-Type: application/x-pki-message

<binary CMS data>
```

When implemented using HTTP GET, this might look as follows:

```
GET /cgi-bin/pkiclient.exe?operation=PKIOperation& \
message=MIAGCSqGSIb3DQEHA6CAMIACAQAxgDCBzAIBADB2MG \
IxETAPBgNVBAcTCE......AAAAAA== HTTP/1.1
```

4.3.1. Certificate Enrolment/Renewal Response Message

If the request is granted, a CertRep SUCCESS message (Section 3.3.2.1) is returned. If the request is rejected, a CertRep FAILURE message (Section 3.3.2.2) is returned. If the CA is configured to manually authenticate the client, a CertRep PENDING message (Section 3.3.2.3) **MAY** be returned. The CA **MAY** return a PENDING for other reasons.

The response will have a Content-Type of "application/x-pki-message".

```
"Content-Type: application/x-pki-message"

<binary CertRep message>
```

4.4. Poll for Client Initial Certificate

When the client receives a CertRep message with pkiStatus set to PENDING, it will enter the polling state by periodically sending CertPoll messages to the CA until either the request is granted and the certificate is sent back or the request is rejected or some preconfigured time limit for polling or maximum number of polls is exceeded. The OPERATION **MUST** be set to "PKIOperation".

CertPoll messages exchanged during the polling period **MUST** carry the same transactionID attribute as the previous PKCSReq/RenewalReq. A CA receiving a CertPoll for which it does not have a matching PKCSReq/RenewalReq **MUST** reject this request.

Since at this time the certificate has not been issued, the client can only use its own subject name (which was contained in the original PKCS# 10 sent via PKCSReq/RenewalReq) to identify the polled certificate request (but see the note on identification during polling in Section 3.3.3). In theory, there can be multiple outstanding requests from one client (for example, if different keys and different key usages were used to request multiple certificates), so the transactionID must also be included to disambiguate between multiple requests. In practice, however, the client **SHOULD NOT** have multiple requests outstanding at any one time, since this tends to confuse some CAs.

4.4.1. Polling Response Message Format

The response messages for CertPoll are the same as in Section 4.3.1.

4.5. Certificate Access

A client can query an issued certificate from the SCEP CA, as long as the client knows the issuer name and the issuer-assigned certificate serial number.

This transaction consists of one GetCert (Section 3.3.4) message sent to the CA by a client and one CertRep (Section 3.3.2) message sent back from the CA. The OPERATION **MUST** be set to "PKIOperation".

4.5.1. Certificate Access Response Message Format

In this case, the CertRep from the CA is same as in Section 4.3.1, except that the CA will either grant the request (SUCCESS) or reject it (FAILURE).

4.6. CRL Access

Clients can request a CRL from the SCEP CA, as described in Section 2.7. The OPERATION **MUST** be set to "PKIOperation".

4.6.1. CRL Access Response Message Format

The CRL is sent back to the client in a CertRep (Section 3.3.2) message. The information portion of this message is a degenerate certificates-only SignedData (Section 3.4) that contains only the most recent CRL in the crls field of the SignedData.

4.7. Get Next Certificate Authority Certificate

When a CA certificate is about to expire, clients need to retrieve the CA's next CA certificate (i.e., the rollover certificate). This is done via the GetNextCACert message. The OPERATION **MUST** be set to "GetNextCACert". There is no request data associated with this message.

4.7.1. Get Next CA Response Message Format

The response consists of a SignedData CMS message, signed by the current CA signing key. Clients **MUST** validate the signature on the message before trusting any of its contents. The response will have a Content-Type of "application/x-x509-next-ca-cert".

```
"Content-Type: application/x-x509-next-ca-cert"

<binary CMS>
```

The content of the SignedData message is a degenerate certificates-only SignedData message (Section 3.4) containing the new CA certificate(s) to be used when the current CA certificate expires.

5. SCEP Transaction Examples

The following section gives several examples of client-to-CA transactions. Client actions are indicated in the left column, CA actions are indicated in the right column, and the transactionID is given in parentheses. For ease of reading, small integer values have been used; in practice, full transaction IDs would be used. The first transaction, for example, would read like this:

> Client Sends PKCSReq message with transactionID 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 certificate locally.

5.1. Successful Transactions

```
PKCSReq (1)             ----------> CA issues certificate
                        <---------- CertRep (1) SUCCESS
Client installs certificate
```

Figure 7: Successful Enrolment Case: Automatic Processing

```
PKCSReq (2)             ----------> Cert request goes into queue
                        <---------- CertRep (2) PENDING
CertPoll (2)            ----------> Still pending
                        <---------- CertRep (2) PENDING
CertPoll (2)            ----------> CA issues certificate
                        <---------- CertRep (2) SUCCESS
Client installs certificate
```

Figure 8: Successful Enrolment Case: Manual Authentication Required

```
GetNextCACert         ---------->
                      <---------- New CA certificate

PKCSReq*              ----------> CA issues certificate with
                                  new key
                      <---------- CertRep SUCCESS
Client stores certificate
for installation when
existing certificate expires.
```

Figure 9: CA Certificate Rollover Case

* Enveloped for the new CA certificate. The CA will use the envelope to determine which key to use to issue the client certificate.

5.2. Transactions with Errors

In the case of polled transactions that aren't completed automatically, there are two potential options for dealing with a transaction that's interrupted due to network or software/hardware issues. The first is for the client to preserve its transaction state and resume the CertPoll polling when normal service is restored. The second is for the client to begin a new transaction by sending a new PKCSReq/RenewalReq, rather than continuing the previous CertPoll. Both options have their own advantages and disadvantages.

The CertPoll continuation requires that the client maintain its transaction state for the time when it resumes polling. This is relatively simple if the problem is a brief network outage, but less simple when the problem is a client crash and restart. In addition, the CA may treat a lost network connection as the end of a transaction, so that a new connection followed by a CertPoll will be treated as an error.

The PKCSReq/RenewalReq continuation doesn't require any state to be maintained, since it's a new transaction. However, it may cause problems on the CA side if the certificate was successfully issued but the client never received it, since the resumed transaction attempt will appear to be a request for a duplicate certificate (see Section 7.4 for more on why this is a problem). In this case, the CA may refuse the transaction or require manual intervention to remove/revoke the previous certificate before the client can request another one.

Since the new-transaction resume is more robust in the presence of errors and doesn't require special-case handling by either the client or CA, clients **SHOULD** use the new-transaction option in preference to the resumed-CertPoll option to recover from errors.

Resync Case 1: Client resyncs via new PKCSReq (recommended):

```
PKCSReq (3)          ----------> Cert request goes into queue
                     <---------- CertRep (3) PENDING
CertPoll (3)         ----------> Still pending
                       X-------- CertRep(3) PENDING
(Network outage)
(Client reconnects)
PKCSReq (4)          ---------->
                     <---------- CertRep (4) PENDING
etc...
```

Figure 10: Resync Case 1

Resync Case 2: Client resyncs via resumed CertPoll after a network outage (not recommended; use PKCSReq to resync):

```
PKCSReq (5)          ----------> Cert request goes into queue
                     <---------- CertRep (5) PENDING
CertPoll (5)         ----------> Still pending
                       X-------- CertRep(5) PENDING
(Network outage)
(Client reconnects)
CertPoll (5)         ----------> CA issues certificate
                     <---------- CertRep (5) SUCCESS
Client installs certificate
```

Figure 11: Resync Case 2

Resync Case 3: Special-case variation of Case 2 where the CertRep SUCCESS rather than the CertRep PENDING is lost (recommended):

```
PKCSReq (6)          ----------> Cert request goes into queue
                     <---------- CertRep (6) PENDING
CertPoll (6)         ----------> Still pending
                     <---------- CertRep (6) PENDING
CertPoll (6)         ----------> CA issues certificate
                       X-------- CertRep(6) SUCCESS
(Network outage)
(Client reconnects)
PKCSReq (7)          ----------> There is already a valid
                                 certificate with this
                                 Distinguished Name (DN).
                     <---------- CertRep (7) FAILURE
                                 Admin revokes certificate
PKCSReq (8)          ----------> CA issues new certificate
                     <---------- CertRep (8) SUCCESS
Client installs certificate
```

Figure 12: Resync Case 3

Resync Case 4: Special-case variation of Case 1 where the CertRep SUCCESS rather than the CertRep PENDING is lost (not recommended; use PKCSReq to resync):

```
PKCSReq (9)           ----------> Cert request goes into queue
                      <---------- CertRep (9) PENDING
CertPoll (9)          ----------> Still pending
                      <---------- CertRep (9) PENDING
CertPoll (9)          ----------> CA issues certificate
                        X-------- CertRep(9) SIGNED CERT
(Network outage)
(Client reconnects)
CertPoll (9)          ----------> Certificate already issued
                      <---------- CertRep (9) SUCCESS
Client installs certificate
```

Figure 13: Resync Case 4

As these examples indicate, resumption from an error via a resumed CertPoll is tricky due to the state that needs to be held by both the client and/or the CA. A PKCSReq/RenewalReq resume is the easiest to implement, since it's stateless and is identical for both polled and nonpolled transactions, whereas a CertPoll resume treats the two differently. (A nonpolled transaction is resumed with a PKCSReq/RenewalReq; a polled transaction is resumed with a CertPoll.) For this reason, error recovery **SHOULD** be handled via a new PKCSReq rather than a resumed CertPoll.

6. IANA Considerations

An object identifier for an arc to assign SCEP Attribute Identifiers has been assigned in the "SMI Security for PKIX" registry (1.3.6.1.5.5.7). This object identifer, Simple Certificate Enrollment Protocol Attributes, is denoted as id-scep:

```
id-scep OBJECT IDENTIFIER ::= { id-pkix 24 }
```

IANA created the "SMI Security for SCEP Attribute Identifiers" registry (1.3.6.1.5.5.7.24) with the following entries with references to this document:

```
id-scep-failInfoText OBJECT IDENTIFIER ::= { id-scep 1 }
```

Entries in the registry are assigned according to the "Specification Required" policy defined in [RFC8126].

Section 3.2.1.2 describes an "SCEP Message Type" registry, and Section 3.5 describes an "SCEP CA Capabilities" registry; these registries are maintained by IANA and define a number of such code-point identifiers. Entries in the registry are assigned according to the "Specification Required" policy defined in [RFC8126].

The "SCEP Message Types" registry has "Value", "Name", "Description", and "Reference" columns. The "Value" entry is a small positive integer; value "0" is reserved.

The "SCEP CA Capabilities" registry has "Keyword", "Description", and "Reference" columns. Although implementations **SHOULD** use the "SCEP CA Capabilities" registry, SCEP is often employed in situations where this isn't possible. In this case, private-use CA capabilities may be specified using a unique prefix such as an organisation identifier or domain name under the control of the entity that defines the capability. For example, the prefix would be "Example.com-", and the complete capability would be "Example.com-CapabilityName".

IANA has registered four media types as defined in this document:

- application/x-x509-ca-cert
- application/x-x509-ca-ra-cert
- application/x-x509-next-ca-cert
- application/x-pki-message

Note that these are grandfathered media types registered as per Appendix A of [RFC6838]. Templates for registrations are specified below.

6.1. Registration of the application/x-x509-ca-cert Media Type

Type name: application

Subtype name: x-x509-ca-cert

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations: This media type contains a certificate; see the Security Considerations section of [RFC5280]. There is no executable content.

Interoperability considerations: This is a grandfathered registration of an alias to application/pkix-cert (basically a single DER-encoded Certification Authority certificate), which is only used in SCEP.

Published specification: RFC 8894

Applications that use this media type: SCEP uses this media type when returning a CA certificate.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): none

File extension(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information: See the Authors' Addresses section of RFC 8894.

Intended usage: LIMITED USE

Restrictions on usage: SCEP protocol

Author: See the Authors' Addresses section of RFC 8894

Change controller: IETF

Provisional registration? No

6.2. Registration of the application/x-x509-ca-ra-cert Media Type

Type name: application

Subtype name: x-x509-ca-ra-cert

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations: This media type consists of a degenerate certificates-only CMS SignedData message (Section 3.4) containing the certificates, with the intermediate CA certificate(s) as the leaf certificate(s). There is no executable content.

Interoperability considerations: This is a grandfathered registration that is only used in SCEP.

Published specification: RFC 8894

Applications that use this media type: SCEP uses this media type when returning CA Certificate Chain Response.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): none

File extension(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information: See the Authors' Addresses section of RFC 8894.

Intended usage: LIMITED USE

Restrictions on usage: SCEP protocol

Author: See the Authors' Addresses section of RFC 8894.

Change controller: IETF

Provisional registration? no

6.3. Registration of the application/x-x509-next-ca-cert Media Type

Type name: application

Subtype name: x-x509-next-ca-cert

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations: This media type consists of a SignedData CMS message, signed by the current CA signing key. There is no executable content.

Interoperability considerations: This is a grandfathered registration that is only used in SCEP.

Published specification: RFC 8894

Applications that use this media type: SCEP uses this media type when returning a Get Next CA response.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): none

File extension(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information: See the Authors' Addresses section of RFC 8894.

Intended usage: LIMITED USE

Restrictions on usage: SCEP protocol

Author: See the Authors' Addresses section of RFC 8894.

Change controller: IETF

Provisional registration? no

6.4. Registration of the application/x-pki-message Media Type

Type name: application

Subtype name: x-pki-message

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations: This media type consists of a degenerate certificates-only CMS SignedData message. There is no executable content.

Interoperability considerations: This is a grandfathered registration that is only used in SCEP.

Published specification: RFC 8894

Applications that use this media type: SCEP uses this media type when returning a Certificate Enrolment/Renewal Response.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): none

File extension(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information: See the Authors' Addresses section of RFC 8894.

Intended usage: LIMITED USE

Restrictions on usage: SCEP protocol

Author: See the Authors' Addresses section of RFC 8894.

Change controller: IETF

Provisional registration? no

7. Security Considerations

The security goal of SCEP is that no adversary can subvert the public key/identity binding from that intended. An adversary is any entity other than the client and the CA participating in the protocol.

This goal is met through the use of CMS and PKCS #10 encryption and digital signatures using authenticated public keys. The CA's public key is authenticated via out-of-band means such as the checking of the CA fingerprint, and the SCEP client's public key is authenticated through manual or preshared secret authentication.

7.1. General Security

Common key-management 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. This is especially true for CA keys which, when compromised, compromise the security of all relying parties.

7.2. Use of the CA Private Key

A CA private key is generally meant for, and usually flagged as, being usable for certificate (and CRL) signing exclusively rather than data signing or encryption. The SCEP protocol, however, uses the CA private key to both sign and optionally encrypt CMS transport messages. This is generally considered undesirable, as it widens the possibility of an implementation weakness and provides an additional location where the private key must be used (and hence is slightly more vulnerable to exposure) and where a side-channel attack might be applied.

7.3. ChallengePassword Shared Secret Value

The security measures that should be applied to the challengePassword shared secret depend on the manner in which SCEP is employed. In the simplest case, with SCEP used to provision devices with certificates in the manufacturing facility, the physical security of the facility may be enough to protect the certificate issue process with no additional measures explicitly required. In general, though, the security of the issue process depends on the security employed around the use of the challengePassword shared secret. While it's not possible to enumerate every situation in which SCEP may be utilised, the following security measures should be considered.

- The challengePassword, despite its name, shouldn't be a conventional password but a high-entropy shared-secret authentication string. Using the base64 encoding of a keying value generated or exchanged as part of standard device authentication protocols like the Extensible Authentication Protocol (EAP) or DNP3 Secure Authentication (DNP3-SA) makes for a good challengePassword. The use of high-entropy shared secrets is particularly important when the PasswordRecipientInfo option is used to encrypt SCEP messages; see Section 3.1.

- If feasible, the challengePassword should be a one-time value used to authenticate the issue of a single certificate (subsequent certificate requests will be authenticated by being signed with the initial certificate). If the challengePassword is single use, then the arrival of subsequent requests using the same challengePassword can then be used to indicate a security breach.
- The lifetime of a challengePassword can be limited, so that it can be used during initial device provisioning but will have expired at a later date if an attacker manages to compromise the challengePassword value -- for example, by compromising the device that it's stored in.
- The CA should take appropriate measures to protect the challengePassword. Examples of possible measures include: physical security measures; storing it as a salted iterated hash or equivalent memory-hard function; storing it as a keyed MAC value if it's not being used for encryption; and storing it in encrypted form if it is being used for encryption.

7.4. Lack of Certificate Issue Confirmation

SCEP provides no confirmation that the issued certificate was successfully received and processed by the client. This means that if the CertRep message is lost or can't be processed by the client, then the CA will consider the certificate successfully issued while the client won't. If this situation is of concern, then the correct issuance of the certificate will need to be verified by out-of-band means, for example, through the client sending a message signed by the newly issued certificate to the CA. This also provides the proof of possession that's not present in the case of a renewal operation; see Section 7.6.

7.5. GetCACaps Issues

The GetCACaps response is not authenticated by the CA. This allows an attacker to perform downgrade attacks on the cryptographic capabilities of the client/CA exchange. In particular, if the server were to support MD5 and single DES, then an in-path attacker could trivially roll back the encryption to use these insecure algorithms. By taking advantage of the presence of large amounts of static known plaintext in the SCEP messages, as of 2017, a DES rainbow table attack can recover most encryption keys in under a minute, and MD5 chosen-prefix collisions can be calculated for a few tens of cents of computing time using tools like HashClash. It is for this reason that this specification makes single DES and MD5 a **MUST NOT** feature. Note that all known servers support at least triple DES and SHA-1 (regardless of whether "DES3" and "SHA-1" are indicated in GetCACaps), so there should never be a reason to fall all the way back to single DES and MD5.

One simple countermeasure to a GetCACaps downgrade attack is for clients that are operating in an environment where on-path attacks are possible and that expect the "SCEPStandard" capability to be indicated by the CA but don't see it in the GetCACaps response to treat its absence as a security issue, and either discontinue the exchange or continue as if "SCEPStandard" had been returned. This requires a certain trade-off between compatibility with old servers and security against active attacks.

7.6. Lack of PoP in Renewal Requests

Renewal operations (but not standard certificate-issue operations) are processed via a previously issued certificate and its associated private key, not the key in the PKCS #10 request. This means that a client no longer demonstrates proof of possession (PoP) of the private key corresponding to the public key in the PKCS #10 request. It is therefore possible for a client to recertify an existing key used by a third party, so that two or more certificates exist for the same key. By switching out the certificate in a signature, an attacker can appear to have a piece of data signed by their certificate rather than the original signer's certificate. This, and other, attacks are described in S/MIME ESS [RFC2634].

Avoiding these types of attacks requires situation-specific measures. For example, CMS/SMIME implementations may use the ESSCertID attribute from S/MIME ESS [RFC2634] or its successor, S/MIME ESSv2 [RFC5035], to unambiguously identify the signing certificate. However, since other mechanisms and protocols that the certificates will be used with typically don't defend against this problem, it's unclear whether this is an actual issue with SCEP.

7.7. Traffic Monitoring

SCEP messages are signed with certificates that may contain identifying information. If these are sent over the public Internet and real identity information (rather than placeholder values or arbitrary device IDs) is included in the signing certificate data, an attacker may be able to monitor the identities of the entities submitting the certificate requests. If this is an issue, then [RFC7258] should be consulted for guidance.

7.8. Unnecessary Cryptography

Some of the SCEP exchanges use unnecessary signing and encryption operations. In particular, the GetCert and GetCRL exchanges are encrypted and signed in both directions. The information requested is public, and thus encrypting the requests is of questionable value. In addition, CRLs and certificates sent in responses are already signed by the CA and can be verified by the recipient without requiring additional signing and encryption. More lightweight means of retrieving certificates and CRLs such as HTTP certificate-store access [RFC4387] and LDAP are recommended for this reason.

7.9. Use of SHA-1

The majority of the large number of devices that use SCEP today default to SHA-1, with many supporting only that hash algorithm with no ability to upgrade to a newer one. SHA-1 is no longer regarded as secure in all situations, but as used in SCEP, it's still safe. There are three reasons for this. The first is that attacking SCEP would require creating a fully general SHA-1 collision in close to real time alongside breaking AES (more specifically, it would require creating a fully general SHA-1 collision for the PKCS #10 request, breaking the AES encryption around the PKCS #10 request, and then creating a second SHA-1 collision for the signature on the encrypted data), which won't be feasible for a long time.

The second reason is that the signature over the message -- in other words, the SHA-1 hash that isn't protected by encryption -- doesn't serve any critical cryptographic purpose: The PKCS #10 data itself is authenticated through its own signature, protected by encryption, and the overall request is authorised by the (encrypted) shared secret. The sole exception to this will be the small number of implementations that support the Renewal operation, which may be authorised purely through a signature, but presumably any implementation recent enough to support Renewal also supports SHA-2. Any legacy implementation that supports the historic core SCEP protocol would not be affected.

The third reason is that SCEP uses the same key for encryption and signing, so that even if an attacker were able to capture an outgoing renewal request that didn't include a shared secret (in other words, one that was only authorised through a signature), break the AES encryption, forge the SHA-1 hash in real time, and forward the forged request to the CA, they couldn't decrypt the returned certificate, which is protected with the same key that was used to generate the signature. While Section 7.8 points out that SCEP uses unnecessary cryptography in places, the additional level of security provided by the extra crypto makes it immune to any issues with SHA-1.

This doesn't mean that SCEP implementations should continue to use SHA-1 in perpetuity, merely that there's no need for a panicked switch to SHA-2.

7.10. Use of HTTP

SCEP is an encrypted, authenticated certificate enrollment protocol that uses HTTP as a simple transport mechanism. Since SCEP messages are already cryptographically secured, it does not require transport layer security. Where HTTPS is elected, a performance hit may result from the TLS overhead, operational problems may result due to the more complex configuration, and potential security vulnerability may result due to the addition of an entire TLS protocol stack alongside the basic SCEP protocol.

In particular, experience has shown that the issue of configuring certificates, CAs, and trust for both TLS and SCEP often leads to interoperability problems because different certificates and trust models are used in each. Use of HTTPS to authenticate the server does not enable omission of the ChallengePassword or similar authenticator in the SCEP message on the assumption that using HTTPS instead of HTTP will somehow make this insecure usage secure again. HTTPS is not soy sauce for security and is unnecessary for SCEP, which uses cryptographically secured messages and does not require transport layer security.

8. References

8.1. Normative References

[AES] Technology, U. N. I. O. S. A., "The Advanced Encryption Standard (AES)", FIPS 197, DOI 10.6028/NIST.FIPS.197, November 2001, <https://doi.org/10.6028/NIST.FIPS.197>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, DOI 10.17487/RFC2985, November 2000, <https://www.rfc-editor.org/info/rfc2985>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/info/rfc2986>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SHA2] Technology, U. N. I. O. S. A., "Secure Hash Standard (SHS)", FIPS 180-3, October 2008.

8.2. Informative References

[HTTP] Nottingham, M., "Building Protocols with HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-bcp56bis-09, November 1, 2019, <https://tools.ietf.org/html/draft-ietf-httpbis-bcp56bis-09>.

[JSCEP] "A Java implementation of the Simple Certificate Enrolment Protocol", commit 7410332, January 2020, <https://github.com/jscep/jscep>.

[RFC2634] Hoffman, P., Ed., "Enhanced Security Services for S/MIME", RFC 2634, DOI 10.17487/RFC2634, June 1999, <https://www.rfc-editor.org/info/rfc2634>.

[RFC4387] Gutmann, P., Ed., "Internet X.509 Public Key Infrastructure Operational Protocols: Certificate Store Access via HTTP", RFC 4387, DOI 10.17487/RFC4387, February 2006, <https://www.rfc-editor.org/info/rfc4387>.

[RFC5035] Schaad, J., "Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility", RFC 5035, DOI 10.17487/RFC5035, August 2007, <https://www.rfc-editor.org/info/rfc5035>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

Appendix A. Background Notes

This specification has spent over twenty years in the draft stage. Its original goal, provisioning IPsec routers with certificates, has long since changed to general device/embedded system/IoT use. To fit this role, extra features were bolted on in a haphazard manner through the addition of a growing list of appendices and by inserting additional, often conflicting, paragraphs in various locations in the body text. Since existing features were never updated as newer ones were added, the specification accumulated large amounts of historical baggage over time. If OpenPGP was described as "a museum of 1990s crypto", then the SCEP document was its graveyard.

About five years ago, the specification, which even at that point had seen only sporadic reposts of the existing document, was more or less abandoned by its original sponsors. Due to its widespread use in large segments of the industry, the specification was rebooted in 2015, cleaning up fifteen years' worth of accumulated cruft, fixing errors, clarifying ambiguities, and bringing the algorithms and standards used into the current century (prior to the update, the de facto lowest-common-denominator algorithms used for interoperability were the insecure forty-year-old single DES and broken MD5 hash algorithms).

Note that although the text of the current specification has changed significantly due to the consolidation of features and appendices into the main document, the protocol that it describes is identical on the wire to the original (with the unavoidable exception of the switch from single DES and MD5 to AES and SHA-2). The only two changes introduced, the "SCEPStandard" indicator in GetCACaps and the failInfoText attribute, are both optional values and would be ignored by older implementations that don't support them, or can be omitted from messages if they are found to cause problems.

Other changes include:

- Resolved contradictions in the text -- for example, a requirement given as a **MUST** in one paragraph and a **SHOULD** in the next, a **MUST NOT** in one paragraph and a **MAY** a few paragraphs later, a **SHOULD NOT** contradicted later by a **MAY**, and so on.
- Merged several later fragmentary addenda placed in appendices (for example, the handling of certificate renewal) with the body of the text.
- Merged the "SCEP Transactions" and "SCEP Transport" sections, since the latter mostly duplicated (with occasional inconsistencies) the former.
- Updated the algorithms to ones dating from at least this century.
- Did the same for normative references to other standards.
- Updated the text to use consistent terminology for the client and CA rather than a mixture of client, requester, requesting system, end entity, server, certificate authority, certification authority, and CA.
- Corrected incorrect references to other standards, e.g., IssuerAndSerial -> IssuerAndSerialNumber.
- Corrected errors such as a statement that when both signature and encryption certificates existed, the signature certificate was used for encryption.
- Condensed redundant discussions of the same topic spread across multiple sections into a single location. For example, the description of intermediate CA handling previously existed in three different locations, with slightly different requirements in each one.
- Added a description of how pkiMessages were processed, which was never made explicit in the original specification. This led to creative interpretations that had security problems but were employed anyway due to the lack of specific guidance on what to do.
- Relaxed some requirements that didn't serve any obvious purpose and that major implementations didn't seem to be enforcing. For example, the requirement that the self-signed certificate used with a request **MUST** contain a subject name that matched the one in the PKCS #10 request was relaxed to a **SHOULD**, because a number of implementations either ignored the issue entirely or at worst performed some minor action like creating a log entry, after which they continued anyway.

- Removed discussion of the transactionID from the security considerations, since the instructions there were directly contradicted by the discussion of the use of the transactionID in Section 5.
- Added a requirement that the signed message include the signing certificate(s) in the signedData certificates field. This was implicit in the original specification (without it, the message couldn't be verified by the CA) and was handled by the fact that most PKCS #7/CMS libraries do this by default, but was never explicitly mentioned.
- Clarified sections that were unclear or even made no sense -- for example, the requirement for a "hash on the public key" [sic] encoded as a PrintableString.
- Renamed "RA certificates" to "intermediate CA certificates". The original document at some point added mention of RA certificates without specifying how the client was to determine that an RA was in use, how the RA operations were identified in the protocol, or how it was used. It's unclear whether what was meant was a true RA or merely an intermediate CA, as opposed to the default practice of having certificates issued directly from a single root CA certificate. This update uses the term "intermediate CA certificates", since this seems to have been the original intent of the text.
- Redid the PKIMessage diagram to match what was specified in CMS; the original diagram omitted a number of fields and nested data structures, which meant that the diagram didn't match either the text or the CMS specification.
- Removed the requirement for a CertPoll to contain a recipientNonce, since CertPoll is a client message and will never be sent in response to a message containing a senderNonce. See also the note in Section 3.3.2.
- Clarified certificate renewal. This represents a capability that was bolted onto the original protocol with (at best) vaguely defined semantics, including a requirement by the CA to guess whether a particular request was a renewal or not. In response to developer feedback that they either avoided renewal entirely because of this uncertainty or hard-coded in particular behaviour on a per-CA basis, this specification explicitly identifies renewal requests as such and provides proper semantics for them.
- Corrected the requirement that "undefined message types are treated as an error", since this negates the effect of GetCACaps, which is used to define new message types. In particular, operations such as GetCACaps "Renewal" would be impossible if enforced as written, because the Renewal operation was an undefined message type at the time.
- In line with the above, added IANA registries for several entries that had previously been defined in an ad hoc manner in different locations in the text.
- Added the "SCEPStandard" keyword to GetCACaps to indicate that the CA complies with the final version of the SCEP standard, since the definition of what constitutes SCEP standards compliance has changed significantly over the years.
- Added the optional failInfoText attribute to deal with the fact that failInfo was incapable of adequately communicating to clients why a certificate request operation had been rejected.
- Removed the discussion in the security considerations of revocation issues, since SCEP doesn't support revocation as part of the protocol.
- Clarified the use of nonces, which if applied as originally specified would have made the use of polling in the presence of a lost message impossible.

- Removed the discussion of generating a given transactionID by hashing the public key, since this implied that there was some special significance in the value generated this way. Since it was neither a **MUST** nor a **MAY**, it was unsound to imply that servers could rely on the value being generated a certain way. In addition, it wouldn't work if multiple transactions as discussed in Section 4.4 were initiated, since the deterministic generation via hashing would lead to duplicate transactionIDs.
- Added examples of SCEP messages to give implementers something to aim for.

Acknowledgements

The editor would like to thank all of the previous editors, authors, and contributors for their work maintaining the document over the years: Cheryl Madson, Xiaoyi Liu, David McGrew, David Cooper, Andy Nourse, Max Pritikin, Jan Vilhuber, and others. The IETF reviewers provided much useful feedback that helped improve the document, and in particular spotted a number of things that were present in SCEP through established practice rather than by being explicitly described in the text. Numerous other people have contributed during the long life cycle of the document, and all deserve thanks. In addition, several PKCS #7 / CMS libraries contributed to interoperability by doing the right thing despite what earlier SCEP documents required.

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

Author's Address

Peter Gutmann
University of Auckland
Department of Computer Science
Auckland
New Zealand
Email: pgut001@cs.auckland.ac.nz