```
Network Working Group                                          A. Kapoor
Internet-Draft                                              R. Tschalaer
Updates: 2510,4210                                              Certicom
(if approved)                                                  T. Kause
Intended status: Standards Track                                    SSH
Expires: October 29, 2010                                      M. Peylo
                                                                    NSN
                                                         April 27, 2010
```

Internet X.509 Public Key Infrastructure -- Transport Protocols for CMP
draft-ietf-pkix-cmp-transport-protocols-08.txt

Abstract

This document describes how to layer Certificate Management Protocols over various transport protocols.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The Certificate Management Protocol (CMP) [RFC4210] requires well defined transport mechanisms to enable End Entities, RAs and CAs to pass PKIMessage sequences between them. This document defines the transport mechanisms which were removed from the main CMP specification with the second release and referred to be in a separate document.

The first version of the CMP specification [RFC2510] included a brief description of a simple TCP-based transport protocol. Its features are simple transport level error-handling and a mechanism to poll for outstanding PKI messages. Additionally, it was mentioned that PKI messages could also be conveyed using file-, E-mail- and HTTP-based transport.

The current version of the CMP specification incorporated an own polling mechanism and thus the need for a transport protocol providing this functionality vanished. The remaining features CMP requires from its transport protocols are connection- and error-handling.

During the long time it existed as draft, this RFC was undergoing drastic changes. The TCP-based transport specification was enhanced and a TCP-Messages-over-HTTP transport specification appeared. Both proved to be needless and cumbersome, implementers preferred to use plain HTTP transport. This specification now aims to reflect that.

HTTP transport is generally easy to implement, traverses network borders utilizing ubiquitous proxies and is already commonly found in existing implementations. TCP-based transport is only documented for information and optional downward compatibility. E-Mail or file transfer are also mentioned and may be used to convey PKIMessage sequences - provided that scenarios are identified where they are better suited than HTTP.

2. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. TCP-Based Management Protocol

The so-called "TCP-based transport" is OPTIONAL and its use is deprecated. Its description appears here only for information and downward compatibility. HTTP-based transport, as described in Section 4, SHOULD be preferred when transporting CMP messages as defined in [RFC4210]. The reasoning for that is given in Section 1.

While this section is called TCP-based and the messages are called TCP-Messages, the same protocol can be used over any reliable, connection oriented transport protocol (e.g. SNA, DECnet, etc.). This protocol is suitable for cases where an end entity (or an RA) initiates a transaction and can poll to pick up the results.

The client sends a TCP-Message to the server, and the server responds with another TCP-Message. A response MUST be sent for every request, even if the encapsulated CMP message in the request does not have a corresponding response.

The protocol requires a listener process on an RA or CA which can accept TCP-Messages on a well-defined port (default TCP port number is 829). Typically a client initiates the connection to the server and instantly submits a TCP-Message. The server replies with a TCP-Message containing either a CMP message or a reference number to be used later when polling for the actual CMP response message.

If a polling-reference was supplied, the client SHOULD send a polling request using this polling-reference after waiting for at least the time specified along with the reference number. The server may again reply with a new polling-reference or with the actual CMP message response.

When the final CMP response message has been picked up by the client, no new polling reference is supplied.

3.1. General Form

The format of a TCP-Message is shown below:

```
      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                            Length                             |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |  Version = 10 |     Flags     |  Message-Type |               \
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               /
     \                                                               \
     /                     Value (variable length)                   /
     \                                                               \
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Length: 32 bits (unsigned integer)
    This field contains the number of remaining octets of the TCP-Message (i.e. number of octets of the Value field plus 3). All bit values in this protocol are specified to be in network byte order.

Version: 8-bits (unsigned integer)
    The version of the TCP-Message is 10 in for this document. It MUST be incremented in each future specification modification e.g. changing the Flags field in a way that is not fully backwards compatible.

Flags: 8 bits
    TCP-Message specific flags as described in Section 3.3.

Message-Type: 8 bits
    A value indicating the type of the TCP-Message.

Value: variable length
    Message-type dependent data is stored here. The usage of this field is described along with the respective message-type

3.2. Version

The TCP-Message version is 10 for this document. The number has deliberately been chosen to prevent [RFC2510] compliant applications from treating it as a valid message type. Applications receiving a version less than 10 SHOULD interpret the message as being an [RFC2510] style message.

3.2.1. Version Negotiation

If a client knows the protocol version(s) supported by the server (e.g. from a previous TCP-Message exchange or via some out-of-band means) then it SHOULD send a TCP-Message with the highest version supported both by it and the server. If a client does not know what

version(s) the server supports then it SHOULD send a TCP-Message using the highest version it supports.

If a server receives a TCP-Message version that it supports, then it MUST reply with a TCP-Message of the same version. If the version received is higher than what the server supports, it MUST send back a VersionNotSupported errorMsgRep containing the highest version it supports, see Section 3.4.6.

3.2.2. Detection and Interoperation with RFC2510 Conformant Implementations

Servers wishing to interoperate with clients conforming to [RFC2510] can do so by treating any received message with a version less than 10 as an [RFC2510] message and responding in that format. Servers not wishing to support [RFC2510] messages MUST respond with a [RFC2510] errorMsgRep.

If a client receives a [RFC2510] errorMsgRep (message-type 06) message, it MAY automatically resend the same request on the same connection, falling back to the [RFC2510] format; if the received message is not an errorMsgRep, it MUST terminate the connection. It MAY then retry the communication falling back completely to the [RFC2510] format.

Naturally, a client MUST abort the connection attempt if the server does not support any of the client's supported versions. It SHOULD retry the version negotiation after a delay to check if the server was updated.

3.3. Flags

The LSB of the Flags field is used to indicate a connection close; all other bits in the Flags octet MUST be ignored by receivers, and MUST be set to zero by senders.

3.3.1. Connection Close Flag

By default connections are kept open after the receipt of a response. Either party (client or server) MAY set the connection close bit at any time. If the connection close bit is set on a request, then the server MUST set the bit in the response and close the connection after sending the response. If the bit is set on a response from the server, the client MUST NOT send any further requests on that connection. Applications MAY decide to close an idle connection (one on which no response is outstanding) after some time-out. Because of the problem where a client sends a request and the server closes the connection while the request is still in flight, clients SHOULD

automatically retry a request for which no part of the response could be read due to a connection close or reset.

If the connection is kept open, it MUST only be used for subsequent request/response transactions started by the client - the server MUST NOT use it to send requests to the client. Different transactions may be freely interwoven on the same connection. E.g. a CR/CP need not immediately be followed by the Confirm, but may be followed by any other request from a different transaction.

3.4. Message-Types

Message-Types 0-127 are reserved and are to be issued under IANA auspices. Message-types 128-255 are reserved for application use.

The Message-Types currently defined are:

ID Value	Message Name
'00'H	pkiReq
'01'H	pollRep
'02'H	pollReq
'03'H	finRep
'05'H	pkiRep
'06'H	errorMsgRep

If a server receives an unknown message-type, it MUST reply with an InvalidMessageType errorMsgRep. If a client receives an unknown message-type, it MUST abort the current CMP transaction and terminate the connection.

The different TCP-Message-types are discussed in the following sections:

3.4.1. pkiReq

A pkiReq message conveys a PKIMessage from a client to a server. The Value field of this TCP-Message contains a DER-encoded PKIMessage.

The type of PKIMessages that can be carried by pkiReq TCP-Messages are (in the order they are defined in [RFC4210]):

[0] Initialization Request
[2] Certification Request
[4] PKCS-10 Request
[6] pop Response
[7] Key Update Request

      [9] Key Recovery Request
      [11] Revocation Request
      [13] Cross-Certification Request
      [15] CA Key Update Announcement
      [16] Certificate Announcement
      [17] Revocation Announcement
      [18] CRL Announcement
      [20] Nested Message
      [21] General Message
      [23] Error Message
      [24] Certificate Confirmation
      [25] Polling Request

3.4.2. pkiRep

TCP-Messages of this type are used to send a response to the requestor. The Value field of the pkiRep contains a DER encoded PKIMessage.

The type of PKIMessages that can be carried by such pkiRep messages are (in the order they are defined in [RFC4210]):

      [1] Initialization Response
      [3] Certification Response
      [5] pop Challenge
      [8] Key Update Response
      [10] Key Recovery Response
      [12] Revocation Response
      [14] Cross-Certificate Response
      [19] Confirmation
      [22] General Response
      [23] Error Message
      [26] Polling Response

3.4.3. pollReq

A pollReq is used by a client to check the status of a pending TCP-Message. The Value portion of a pollReq contains:

```
      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                        Polling-Reference                      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Polling-Reference: 32 bits (unsigned integer)
    This polling-reference MUST be the one returned via the
    respective pollRep TCP-Message.

3.4.4. pollRep

A pollRep is sent by the server to the client as response in case there is no PKIMessage ready yet. The Value portion of the pollRep looks as follows:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Polling-Reference                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Time-to-Check-Back                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Polling-Reference: 32 bits (unsigned integer)
    A unique 32-bit number identifying the transaction.

Time-to-Check-Back: 32 bits (unsigned integer)
    The time in seconds indicating the minimum interval after which the client SHOULD check the status again. The duration for which the server keeps the polling-reference unique is left to the implementation.

3.4.5. finRep

A finRep is sent by the server whenever no other response applies, such as after receiving a CMP pkiConf. The Value portion of the finRep SHALL contain:

```
    0 1 2 3 4 5 6 7
   +-+-+-+-+-+-+-+-+
   |     '00'H     |
   +-+-+-+-+-+-+-+-+
```

'00'H: 8 bits
    All bits set to zero.

3.4.6. errorMsgRep

This TCP-Message is sent when a TCP-Message level protocol error is detected. It is imperative that PKIError messages MUST NOT be sent using this message type. Examples of TCP-Message level errors are:

o Invalid protocol version
o Invalid TCP message-type
o Invalid polling reference number

The Value field of the errorMsgRep TCP-Message MUST contain:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Error-Type           |         Data-Length           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   \                                                               \
   /                     Data  (variable length)                   /
   \                                                               \
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Error-Type: 16 bits A value (format described below) indicating the type of the error.

Data-Length: 16 bits (unsigned integer) Contains the length of the Data field in number of octets. Error messages not conveying additional information MUST set Data-Length to 0.

Data: <data-length> octets
An UTF8 text string for user readable error messages, containing additional information about the error. Note that it does not contain a terminating NULL character at the end. It SHOULD include an [RFC5646] language tag, as described in [RFC2482]

The Error-Type is in the format MMNN where M and N are hex digits (0-F) and MM represents the major category and NN the minor. The major categories defined by this specification are:

ID Value	Major Categories
'01'H	TCP-Message version negotiation
'02'H	client errors
'03'H	server errors

The major categories '80'H-'FF'H are reserved for application use.

The different error-types are discussed in the following sections:

3.4.6.1. VersionNotSupported

The VersionNotSupported errorMsgRep is defined as follows:

Field	Value
Error-Type	'0101'H
Data-Length	1
Data	<version>
UTF8-text String	implementation defined

where <version> is the highest TCP-Message protocol version the server supports.

3.4.6.2. GeneralClientError

The GeneralClientError errorMsgRep is defined as follows:

Field	Value
Error-Type	'0200'H
Data-Length	0
Data	<empty>
UTF8-text String	implementation defined

3.4.6.3. InvalidMessageType

The InvalidMessageType errorMsgRep is defined as follows:

Field	Value
Error-Type	'0201'H
Data-Length	1
Data	<message-type>
UTF8-text String	implementation defined

where <message-type> is the invalid Message-Type ID received by the

server.

3.4.6.4. InvalidPollID

The InvalidPollID errorMsgRep is defined as follows:

Field	Value
Error-Type	'0202'H
Data-Length	4
Data	<polling-reference>
UTF8-text String	implementation defined

where <polling-reference> is the polling-reference received by the server, identifying the transaction.

3.4.6.5. GeneralServerError

The GeneralServerError errorMsgRep is defined as follows:

Field	Value
Error-Type	'0300'H
Data-Length	0
Data	<empty>
UTF8-text String	implementation defined

4. HTTP-Based Protocol

For direct interaction between two entities, where a reliable transport protocol like TCP is available, HTTP SHOULD be utilized for conveying CMP messages.

With its status codes, HTTP provides needed error reporting capabilities. General problems on the server side as well as those directly caused by the respective request can be reported to the client.

As CMP implements a transaction ID, identifying transactions consisting of more than just a single request/response pair, the statelessness of HTTP is not blocking its usage as transport protocol for CMP messages.

4.1. HTTP Versions

Either HTTP/1.0 as described in [RFC1945] or HTTP/1.1 as in [RFC2616] SHALL be used. Naturally, the newer version should be preferred. To support legacy implementations, both server and client MUST be able to interact with counterparts utilizing the other HTTP protocol version.

4.2. Persistent Connections

HTTP permits to reuse a connection for subsequent requests. Implementations may use this functionality for messages within the same transaction but MUST NOT rely on that, as e.g. intermediate HTTP proxies might terminate the connection after each request/response pair.

In contrast to HTTP/1.1, persistent connections are explicitly negotiated in HTTP/1.0. To avoid the problems described in chapter 19.6.2 in [RFC2616], HTTP/1.0 implementations must not send Keep-Alive when talking to proxies.

4.3. General Form

An ASN.1 DER-encoded PKIMessage is sent as the entity-body of an HTTP POST request. If this HTTP request is successful, the server returns the CMP reply in the body of the HTTP response. The response status code in this case MUST be 200; other 2xx codes MUST NOT be used for this purpose. The HTTP responses with empty message body to CMP Announcement messages also utilize the status codes 201 and 202 to identify if the information was properly processed.

Note that a server may return any 1xx, 3xx, 4xx, or 5xx status code

if the HTTP request needs further handling or is otherwise not acceptable.

4.4. Media Type

The Internet Media Type "application/pkixcmp" MUST be set in the HTTP header when conveying a PKIMessage.

4.5. Communication Workflow

In CMP most communication is initiated by the end entities where every CMP request triggers a CMP response message from the CA or RA.

The CMP Announcement messages described in Section 4.6.2 are an exception. Their creation may be triggered by events or generated on a regular basis by a CA. The recipient of the Announcement only replies with an HTTP status code acknowledging the receipt or indicating an error but not with a CMP response.

The receipt of every HTTP message is confirmed by the counterpart using HTTP means or it MUST be assumed by the sender that it was not successfully delivered to its destination.

4.6. HTTP Request-URI

The Request-URI is formed as specified in [RFC3986].

4.6.1. Common Client Requests

TODO: The following is not more than a proposal as the exact unified style is currently under discussion.

Client requests containing a PKI message MUST be directed to an Request-URI depicting a directory having a trailing slash. The following list contains all such CMP message types. The prefixed numbers reflect ASN.1 numbering of the respective element.

```
[0] Initialization Request
[2] Certification Request
[4] PKCS-10 Request
[6] pop Response
[7] Key Update Request
[9] Key Recovery Request
[11] Revocation Request
[13] Cross-Certification Request
[15] CA Key Update Announcement
```

[16] Certificate Announcement
[17] Revocation Announcement
[18] CRL Announcement
[20] Nested Message
[21] General Message
[23] Error Message
[24] Certificate Confirmation
[25] Polling Request

An example of a Request-Line and a Host header field in an HTTP/1.1 header, sending a CMP request to a server, located in the "/cmp" directory of the host example.com, would be

```
POST /cmp/ HTTP/1.1
Host: example.com
```

or in the absoluteURI form

```
POST http://example.com/cmp/ HTTP/1.1
Host: example.com
```

As CAs may be logically located either inside the root- or within subdirectories, it is possible to set up multiple, logically separated CAs on one host. If only one CA is present, then the "/cmp" directory should be used as default.

4.6.2. Announcements

A CMP server may create event-triggered announcements or generate them on a regular basis. It MAY also utilize HTTP transport to convey them to a suitable recipient. The ASN.1 encoded structures are sent as the entity-body of an HTTP POST request.

Suitable recipients for CMP announcements might e.g. be repositories storing the announced information such as directory services. Those listen for incoming messages, utilizing the same HTTP Request-URI scheme as defined in Section 4.6.

The following PKIMessages are announcements that may be pushed by a CA. The prefixed numbers reflect ASN.1 numbering of the respective element.

[15] CA Key Update Announcement
[16] Certificate Announcement
[17] Revocation Announcement
[18] CRL Announcement

CMP announcement messages do not require any CMP response. However,

the recipient MUST acknowledge receipt with a HTTP message having an appropriate status code and an empty body. The sending side should assume the delivery unsuccessful without such reply and retry if applicable after waiting for an appropriate time span.

If the announced issue was successfully stored in a database or was already present, the answer MUST be an HTTP message with a "201 Created" status code and empty message body.

In case the announced issue was only stored for further processing, the status code of the returned HTTP message must be "202 Accepted". After an appropriate delay, the server may then try to send the Announcement again and may repeat this until it receives a confirmation that it was successfully stored. The appropriate duration of the delay and the option to increase it between consecutive attempts should be carefully considered.

A receiver MUST answer with suitable 4xx or 5xx HTTP error codes when a problem occurs.

4.6.2.1. CA Key Update Announcement

When updating its key pair, a CA can produce a CA Key Update Announcement Message that can be made available to the relevant end entities.

As an OPTIONAL feature, a CA may also provide this message to be available via an HTTP GET request for the CAKeyUpdAnn.PKI file in the respective CA's path. The query component of the Request-URI contains a string with the ASCII representation of the serialNumber of the certificate holding the old key.

According to [RFC3986], the query component is indicated by the first question mark ("?") character and terminated by a number sign ("#") character or by the end of the URI.

An example of a Request-Line and a Host header field in an HTTP/1.1 header, requesting a CA Key Update Announcement Message for an old certificate with the serialNumber 4711 from a CMP server, located in the "/cmp" directory of the host example.com, would be

```
GET /cmp/CAKeyUpdAnn.PKI?4711 HTTP/1.1
Host: example.com
```

or in the absoluteURI form

```
GET http://example.com/cmp/CAKeyUpdAnn.PKI?4711 HTTP/1.1
Host: example.com
```

If there is no "CA Key Update Announcement" available for the certificate in question, an HTTP "404 Not Found" message MUST be returned.

4.7. HTTP Considerations

In general, CMP messages are not cachable; requests and responses MUST include a "Cache-Control: no-cache" (and, if either side uses HTTP/1.0, a "Pragma: no-cache") to prevent the client from getting cached responses.

Connection management is based on the HTTP provided mechanisms (Connection and Proxy-Connection header fields).

While an implementation MAY make use of all defined features of the HTTP protocol, it SHOULD keep the protocol utilization as simple as possible.

Content codings MAY be applied.

4.8. Compatibility Issues with Legacy Implementations

As this document was subject of multiple changes during the long period of time it was created in, implementations using a different approach for HTTP transport may exist. While only those implementations according to this specification are compliant, implementers should to be aware that there might be existing ones which behave differently.

Legacy implementations might also use an unregistered "application/pkixcmp-poll" MIME type as it was specified in earlier drafts of this document. Here, the entity-body of an HTTP POST request contains a TCP-Message instead of a plain DER-encoded PKIMessage. Effectively, this is conveying PKIMessage over TCP-Message over HTTP.

5. File-Based Protocol

A file containing a PKIMessage MUST contain only the DER encoding of one PKIMessage, there MUST NOT be extraneous header or trailer information in the file.

Such files can be used to transport PKIMessage sequences using e.g. FTP.

6. Mail-Based Protocol

This subsection specifies a means for conveying ASN.1-encoded messages for the protocol exchanges via Internet mail [RFC5321]. A simple MIME object is specified as follows.

```
Content-Type: application/pkixcmp
Content-Transfer-Encoding: base64

<<the ASN.1 DER-encoded PKIX-CMP message, base64-encoded>>
```

This MIME object can be sent and received using common MIME processing engines and provides a simple Internet mail transport for PKIX-CMP messages. Implementations MAY wish to also recognize and use the "application/x-pkixcmp" MIME type (specified in earlier versions of this document) in order to support backward compatibility wherever applicable.

7. Security Considerations

Four aspects need to be considered by server side implementers:

1. There is the risk for denial of service attacks through resource consumption by opening many connections, therefore idle connections should be terminated after an appropriate timeout, maybe also depending on the available free resources. After sending a CMP Error Message, the server should close the connection even if the CMP transaction is not yet fully completed.

2. There is no security at the TCP and HTTP protocol level (unless tunneled via SSL/TLS) and thus information from TCP-Messages or the HTTP protocol SHOULD NOT be used to change state of the transaction. Change of state SHOULD be triggered by the signed PKIMessages which are carried within the TCP-Message.

3. If the server is going to be sending messages with sensitive information (not meant for public consumption) in the clear, it is RECOMMENDED that the server sends back the message directly and not use the TCP-Message pollRep.

4. The TCP-Message polling request/response mechanism can be used for all kinds of denial of service attacks. It is RECOMMENDED that a server does not change the polling-reference between polling requests.

8. Information Security Considerations

CMP provides inbuilt integrity protection and authentication. Due to the nature of a PKI, from a security perspective the information communicated unencrypted does not contain sensitive information.

However, it might be possible for an interceptor to utilize the available information to gather confidential technical or business critical information. Therefore, users of the HTTP CMP transport might want to use HTTP over TLS according to [RFC2818] or should consider to use virtual private networks created e.g. utilizing Internet Protocol Security according to [RFC4301].

9. IANA Considerations

The IANA has already registered TCP and UDP port 829 for "PKIX-3 CA/RA" and the MIME media type "application/pkixcmp" for identifying CMP sequences.

No further action by the IANA is necessary for this document or any anticipated updates.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, September 2005.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

10.2. Informative References

[RFC1945] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

[RFC2482] Whistler, K. and G. Adams, "Language Tagging in Unicode Plain Text", RFC 2482, January 1999.

[RFC2510] Adams, C. and S. Farrell, "Internet X.509 Public Key Infrastructure Certificate Management Protocols", RFC 2510, March 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

Appendix A. Acknowledgments

Up to the fifth draft version of this document, released on November 24th 2000, the sole authors were Amit Kapoor and Ronald Tschlaer from Certicom. Up to this point, besides editorial changes, the TCP-Based transport was described as it is still included herein.

The authors gratefully acknowledge the contributions of various members of the IETF PKIX Working Group and the ICSA CA-talk mailing list (a list solely devoted to discussing CMP interoperability efforts).

By providing ideas, giving hints and doing invaluable review work, the following individuals, listed alphabetically, have significantly contributed to this document:

Peter Gutmann, University of Auckland
Wolf-Dietrich Moeller, Nokia Siemens Networks

Appendix B. Registration of the application/pkixcmp Media Type

To: ietf-types@iana.org
Subject: Registration of MIME media type application/pkixcmp

MIME media type name: application

MIME subtype name: pkixcmp

Required parameters: -

Optional parameters: -

Encoding considerations:

Content may contain arbitrary octet values (the ASN.1 DER encoding of a PKIMessage sequence, as defined in the IETF PKIX Working Group specifications). base64 encoding is required for MIME e-mail; no encoding is necessary for HTTP.

Security considerations:

This MIME type may be used to transport Public-Key Infrastructure (PKI) messages between PKI entities. These messages are defined by the IETF PKIX Working Group and are used to establish and maintain an Internet X.509 PKI. There is no requirement for specific security mechanisms to be applied at this level if the PKI messages themselves are protected as defined in the PKIX specifications.

Interoperability considerations: -

Published specification: this document

Applications which use this media type: Applications using certificate management, operational, or ancillary protocols (as defined by the IETF PKIX Working Group) to send PKI messages via E-Mail or HTTP.

Additional information:

  Magic number (s): -
  File extension (s): ".PKI"
  Macintosh File Type Code (s): -

Person and email address to contact for further information:
Martin Peylo, martin.peylo@nsn.com

Intended usage: COMMON

Author/Change controller: Martin Peylo

Authors' Addresses

Amit Kapoor
Certicom
25801 Industrial Blvd
Hayward, CA
US

Email: amit@trustpoint.com

Ronald Tschalaer
Certicom
25801 Industrial Blvd
Hayward, CA
US

Email: ronald@trustpoint.com

Tomi Kause
SSH Communications Security Corp.
Fredrikinkatu 42
Helsinki 00100
Finland

Email: toka@ssh.com

Martin Peylo
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Email: martin.peylo@nsn.com