

Network Working Group
Request for Comments: 4387
Category: Standards Track

P. Gutmann, Ed.
University of Auckland
February 2006

Internet X.509 Public Key Infrastructure
Operational Protocols: Certificate Store Access via HTTP

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

The protocol conventions described in this document satisfy some of the operational requirements of the Internet Public Key Infrastructure (PKI). This document specifies the conventions for using the Hypertext Transfer Protocol (HTTP/HTTPS) as an interface mechanism to obtain certificates and certificate revocation lists (CRLs) from PKI repositories. Additional mechanisms addressing PKIX operational requirements are specified in separate documents.

Table of Contents

1. Introduction	2
2. HTTP Certificate Store Interface	3
2.1. Converting Binary Blobs into Search Keys	4
2.2. Attribute Types: X.509	5
2.3. Attribute Types: PGP	6
2.4. Attribute Types: XML	6
2.5. Implementation Notes and Rationale	6
2.5.1. Identification	7
2.5.2. Checking of Input Values	9
2.5.3. URI Notes	10
2.5.4. Responses	11
2.5.5. Performance Issues	12
2.5.6. Miscellaneous	13
2.6. Examples	14
3. Locating HTTP Certificate Stores	15
3.1. Information in the Certificate	15
3.2. Use of DNS SRV	16
3.2.1. Example	16
3.3. Use of a "well-known" Location	16
3.3.1. Examples	17
3.4. Manual Configuration of the Client Software	18
3.5. Implementation Notes and Rationale	18
3.5.1. DNS SRV	18
3.5.2. "well-known" Locations	19
3.5.3. Information in the Certificate	19
3.5.4. Miscellaneous	20
4. Security Considerations	20
5. IANA Considerations	22
6. Acknowledgements	22
7. References	22
7.1. Normative References	22
7.2. Informative References	23

1. Introduction

This specification is part of a multi-part standard for the Internet Public Key Infrastructure (PKI) using X.509 certificates and certificate revocation lists (CRLs). This document specifies the conventions for using the Hypertext Transfer Protocol (HTTP), or optionally, HTTPS as an interface mechanism to obtain certificates or public keys, and certificate revocation lists (CRLs), from PKI repositories. Throughout the remainder of this document the generic term HTTP will be used to cover either option.

Although RFC 2585 [RFC2585] covers fetching certificates via HTTP, this merely mentions that certificates may be fetched from a static URL, which doesn't provide any general-purpose interface capabilities to a certificate store. The conventions described in this document allow HTTP to be used as a general-purpose, transparent interface to any type of certificate or key store including flat files, standard databases such as Berkeley DB and relational databases, and traditional X.500/LDAP directories. Typical applications would include use with web-enabled relational databases (which most databases are) or simple {key,value} lookup mechanisms such as Berkeley DB and its various descendants.

Additional mechanisms addressing PKIX operational requirements are specified in separate documents.

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

2. HTTP Certificate Store Interface

The GET method is used in combination with an HTTP query URI [RFC2616] to retrieve certificates from the underlying certificate store:

```
http_URL = "http:" "://" host [ ":" port ] [ abs_path [ "?" query ]]
```

The parameters for the query portion of the URI are a certificate or key identifier consisting of an attribute type and a value that specifies one or more certificates or public keys to be returned from the query:

```
query = attribute '=' value
```

Certificates and public keys are retrieved from one URI (the certificate URI) and CRLs from another URI (the revocation URI). These may or may not correspond to the same certificate store and/or server (the exact interpretation is a local configuration issue). The query value MUST be encoded using the form-urlencoded media type [RFC2854]. Further details of URI construction, size limits, and other factors are given in [RFC2616].

Responses to unsuccessful queries (for example, to indicate a non-match or an error condition) are handled in the standard manner as per [RFC2616]. Clients should in particular be aware that in some instances servers may return HTTP type 3xx redirection requests to explicitly redirect queries to another server. Obviously, implicit DNS-based redirection is also possible.

If more than one certificate matches a query, it MUST be returned as a multipart/mixed response. The returned data MUST be returned verbatim; it MUST NOT use any additional content- or transfer-encoding at the HTTP level (for example, it can't be compressed or encoded as base64 or quoted-printable text). Implementations SHOULD NOT use chunked encoding in responses.

The query component of the URI MAY optionally contain additional attribute/value pairs separated by the standard ampersand delimiter '&' that specify further actions to be taken by the certificate store. Certificate stores SHOULD ignore any additional unrecognised attribute/value pairs present in the URI.

Other information, such as naming conventions and MIME types, is specified in [RFC2585] (with additional MIME types for non-X.509 content in [RFC3156] and [RFC3275]).

2.1. Converting Binary Blobs into Search Keys

Some fields (indicated by the "Process" column in the tables below) are of arbitrary length and/or contain non-textual data. Both of these properties make them unsuited for direct use in HTTP queries. In order to make them usable, fields for which the processing option is "Hash" are first hashed down to a fixed-length 160-bit value. Fields for which the processing option is "Hash" or "Base64" are base64-encoded to transform the binary data into textual forms:

Processing option	Processing step
-------------------	-----------------

"Hash"	Hash the key value using SHA-1 [FIPS180] to produce a 160-bit value, then continue with the base64 encoding step that follows.
--------	--

"Hash"	Encode the binary value using base64 encoding to produce a 27-byte text-only value. Base64 encoding of the 20 byte value will produce 28 bytes, and the last byte will always be a '=' padding character. The 27-byte value is created by dropping the trailing '=' character.
"Base64"	

For cases where the binary value is smaller or larger than the 20-byte SHA-1 output (for example, with 64-bit/8 byte PGP key IDs), the final value is created by removing any trailing '=' padding from the encoding of the binary value (this is a generalisation of the above case).

Implementations MUST verify that the base64-encoded values submitted in requests contain only characters in the ranges 'a'-'z', 'A'-'Z', '0'-'9', '+', and '/'. Queries containing any other character MUST be rejected. (See the implementation notes in Section 2.5 and the security considerations in Section 4 for more details on this requirement.)

2.2. Attribute Types: X.509

Permitted attribute types and associated values for use with X.509 certificates and CRLs are described below. Arbitrary-length binary values (as indicated in the table below) are converted into a search key by the process described in Section 2.1. Note that the values are checked for an exact match (after decoding of any form-urlencoded [RFC2854] portions if this is necessary) and are therefore case sensitive.

Attribute	Process	Value
-----	-----	-----
certHash	Hash	Search key derived from the SHA-1 hash of the certificate (sometimes called the certificate fingerprint or thumbprint).
uri	None	Subject URI associated with the certificate, without the (optional) scheme specifier. The URI type depends on the certificate. For S/MIME certificates, it would be an email address; for SSL/TLS certificates, it would be the server's DNS name (this is usually also specified as the CommonName); for IPsec certificates, it would be the DNS name/IP address; and so on.
iHash	Hash	Search key derived from the DER-encoded issuer DN as it appears in the certificate, CRL, or other object.
iAndSHash	Hash	Search key derived from the certificate's DER-encoded issuerAndSerialNumber [RFC3852].
name	None	Subject CommonName contained in the certificate.
sHash	Hash	Search key derived from the DER-encoded subject DN as it appears in the certificate or other object.
sKIDHash	Hash	Search key derived from the certificate's subjectKeyIdentifier (specifically the contents octets of the KeyIdentifier OCTET STRING).

Certificate URIs MUST support retrieval by all the above attribute types.

CRL URIs MUST support retrieval by the iHash and sKIDHash attribute types, which identify the issuer of the CRL. In addition, CRL URIs MAY support retrieval by certHash and iAndSHash attribute types, for cases where this is required by the use of the issuingDistributionPoint extension. A CRL query MUST return the matching CRL with the greatest thisUpdate value (in other words, the most recent CRL).

2.3. Attribute Types: PGP

Permitted attribute types and associated values for use with PGP public keys and key revocation information are described below. Binary values (as indicated in the table below) are converted into a search key by the process described in Section 2.1.

Attribute	Process	Value
-----	-----	-----
email	None	email address associated with the key.
fingerprint	Base64	160-bit PGP key fingerprint [RFC2440].
keyID	Base64	64-bit PGP key ID [RFC2440].
name	None	User name associated with the key.

Key URIs MUST support retrieval by all of the above attribute types.

Revocation URIs MUST support retrieval by the fingerprint and keyID attribute types, which identify the issuer of the key revocation.

2.4. Attribute Types: XML

Permitted attribute types and associated values for use with XML are as specified in sections 2.2 and 2.3. Since XML allows arbitrary attributes to be associated with the <RetrievalMethod> child element of <KeyInfo> [RFC3275], there are no additional special requirements for use with XML.

2.5. Implementation Notes and Rationale

This informative section documents the rationale behind the design in Section 2 and provides guidance for implementors.

2.5.1. Identification

The identifiers are taken from PKCS #15 [PKCS15], a standard that covers (among other things) a transparent interface to a certificate/public key store. These identifiers have been field proven, as they have been in common use for a number of years, typically via PKCS #11 [PKCS11]. Certificate stores and the identifiers that are required for typical certificate lookup operations are analysed in some detail in [Gutmann].

The URI identifier type specifies the identifier associated with the certificate's intended usage with a given Internet security protocol. For example, an SSL/TLS server certificate would contain the server's DNS name (this is traditionally also specified as the CommonName or CN) an S/MIME certificate would contain the subject's email address; an IPsec certificate would contain a DNS name or IP address; and a SIP certificate would contain a SIP URI. A modicum of common sense is assumed when deciding upon an appropriate URI field value.

For historical reasons going back to its primary use as a means of looking up users' S/MIME email certificates, some clients may specify the URI attribute name as "email" rather than "uri". Although not required by this specification, servers may choose to allow the use of "email" as an alias for "uri".

In addition, it is common practice to use the Internet identifier associated with the certificate's intended field of application as the CN for the certificate when this is the most sensible name for the certificate subject. For example, an SSL/TLS server certificate will contain the server's DNS name in the CN field. In web-enabled devices, this may indeed be the only name that exists for the device. It is therefore quite possible that the URI will duplicate the CN, and that it may be the only identifier present (that is, there's no full DN but only a single CN field).

By long-standing convention, URIs in certificates are given without a scheme specifier. For example, an SSL/TLS server certificate would contain `www.example.com` rather than `https://www.example.com`, and an S/MIME certificate would contain `user@example.com` rather than `mailto:user@example.com`. This convention is extended to other URI types as well, so that a certificate containing the (effective) URIs `im:user@example.com` and `xmpp:user@example.com` would be queried using the single URI `user@example.com`. The certificate store would then return all certificates containing this URI, leaving it to the client to determine which one is most appropriate for its use. This approach is taken both because for the most common URI types there's no schema specifier (see the paragraphs above) and no easy way to determine what the intended use is (an SSL/TLS server certificate is

simply one presented by an SSL/TLS server), and because the relying party/client is in a better position to judge the certificate's most appropriate use than the certificate store server.

Another possible identifier that has been suggested is an IP address or DNS name, which will be required for web-enabled embedded devices. This is necessary to allow for example a home automation controller to be queried for certificates for the devices that it controls. Since this value is regarded as the CN for the device, common practice is to use this value for the CN in the same way that web server certificates set the CN to the server's DNS name, so this option is already covered in a widely-accepted manner.

The name and email address are an exact copy of what is present in the certificate, without any canonicalisation or rewriting (other than the transport encoding required by HTTP). This follows standard implementation practice, which transfers an exact copy of these data items in order to avoid problems due to character set translation, handling of whitespace, and other issues.

Hashes are used for arbitrary-length fields such as ones containing DNS in place of the full field to keep the length manageable. In addition, the use of the hashed form emphasizes that searching for structured name data isn't a supported feature, since this is a simple interface to a {key,value} certificate store rather than an HTTP interface to an X.500 directory. Users specifically requiring an HTTP interface to X.500 may use technology such as HTTP LDAP gateways for this purpose.

Although clients will always submit a fixed 160-bit value, servers are free to use as many bits of this value as they require. For example, a server may choose to use only the first 40, 64, 80, or 128 bits for efficiency in searching and maintaining indices.

PGP has traditionally encoded IDs using a C-style 0xABCDDEF notation based on the display format used for IDs in PGP 2.0. Unfortunately, strings in this format are also valid strings in the base64 format, complicated further by the fact that near-misses such as 0xABCDRF could be either a mistyped attempt at a hex ID or a valid base64 ID. For this reason, and to ensure consistency, base64 IDs are used throughout this specification. The search keys used internally will be binary values, so whether these are converted from ASCII-hex or base64 is immaterial in the long run.

The attributes are given shortened name forms (for example, iAndSHash in place of issuerAndSerialNumberHash) in order to keep the lengths reasonable, or common name forms (for example, email in place of

rfc822Name, rfc822Mailbox, emailAddress, mail, or email) where multiple name forms exist.

In some cases, users may require additional, application-specific attribute types. For example, a healthcare application that uses a healthcare ID as the primary key for its databases may require the ability to perform certificate lookups based on this healthcare ID. The formatting and use of such application-specific identifiers is beyond the scope of this document. However, they should begin with 'x-' to ensure that they don't conflict with identifiers that may be defined in future versions of this specification.

2.5.2. Checking of Input Values

The attribute value portion of the identifier should be carefully checked for invalid characters since allowing raw data presents a security risk. Consider, for example, a certificate/public key store implemented using an RDBMS in which the SQL query is built up as "SELECT certificate FROM certificates WHERE iHash = " + <search key>. If <search key> is set to "ABCD;DELETE FROM certificates", the results of the query will be quite different from what was expected by the certificate store administrators. Even a read-only query can be problematic; for example, setting <search key> to "UNION SELECT password FROM master.sysxlogins" will list all passwords in an SQL Server database (in an easily-decrypted format) if the user is running under the sa (DBA) account. For this reason, only valid base64 encodings should be allowed. The same checking applies to queries by name or email address.

Straightforward sanitisation of queries may not be sufficient to prevent all attacks; for example, a filter that removes the SQL query string "DELETE" can be bypassed by submitting the string embedded in another instance of the string. Removing "DELETE" from "DELDELETEETE" leaves the outer "DELETE" in place. Abusing the truncation of over-long strings by filters can also be used as a means of attack, with the attacker ensuring that the truncation occurs in the middle of an escape sequence, bypassing the filtering. Although in theory recursive filtering may help here, the use of parameterised queries (often called placeholders) that aren't vulnerable to SQL injection should be used to avoid these attacks. More information on securing database back-ends may be found in [Birkholz], and more comments on sanitisation and safety concerns may be found in the security considerations section.

2.5.3. URI Notes

Pre-constructed URIs that fetch a certificate/public key matching a fixed search criterion may be useful for items such as web pages or business cards, or even for technical support/helpdesk staff who want to mail to users but can't find the certificate themselves. These URIs may also be used to enforce privacy measures when distributing certificates by perturbing the search key in a manner known only to the certificate/public key store, or to the certificate store and users (in other words, by converting the URI into a capability). For example, a user with a newly-issued certificate could be instructed to fetch it with a key of "x-encrCertHash=...", which is decrypted by the certificate store to fetch the appropriate certificate, ensuring that only the certificate owner can fetch their certificate immediately after issue. Similarly, an organisation that doesn't want to make its certificates available for public query might require a MAC on search keys (e.g., "x-macCertHash=...") to ensure that only authorised users can search for certificates (although a more logical place for access control, if a true web server is being used to access the store, would obviously be at the HTTP level).

The query types have been specifically chosen to be not just an HTTP interface to LDAP but a general-purpose retrieval mechanism that allows arbitrary certificate/public key storage mechanisms (with a bias towards simple {key,value} stores, which are deployed almost universally, whether as ISAM, Berkeley DB, or an RDBMS) to be employed as back-ends. This specification has been deliberately written to be technology neutral, allowing any {key,value} lookup mechanism to be used. It doesn't matter if you choose to have trained chimpanzees look up certificates in books of tables, as long as your method can provide the correct response with reasonable efficiency.

Certificate/public key and CRL stores are allocated separate URIs because they may be implemented using different mechanisms. A certificate store typically contains large numbers of small items, while a CRL store contains a very small number of potentially large items. By providing independent URIs, it's possible to implement the two stores using mechanisms tailored to the data they contain.

PGP combines key and revocation information into a single data object so that it's possible to return both public keys and revocation information from the same URI. If distinct key and revocation servers are available, these can provide a slight performance gain since fetching revocation information doesn't require fetching the key that it applies to. If no separate servers are available, a

single server can be used to satisfy both types of queries with a slight performance loss, since fetching revocation information will also fetch the public key data associated with the revocation data.

2.5.4. Responses

The disallowance of exotic encoding forms reflects the fact that most clients (and many servers, particularly for embedded devices) are not general-purpose web browsers or servers capable of handling an arbitrary range of encoding forms and types, but simply basic HTTP engines attached to key management applications. In other words, the HTTP interface is a rudimentary add-on to a key management application, rather than key-management being an add-on to a general-purpose web client or server. Eliminating unnecessary choices simplifies the implementation task and reduces code size and complexity, with an accompanying decrease in the probability of security issues arising from the added complexity.

The use of an "Accept-encoding: identity" header would achieve the same effect as disallowing any additional encodings and may indeed be useful since section 14.3 of [RFC2616] indicates that the absence of this header may be taken to mean that any encoding is permitted. However, this unnecessarily bloats the HTTP header in a potentially performance-affecting manner (see Section 2.5.5), whereas establishing a requirement that the response be returned without any additional decoration avoids the need to specify this in each request. Implementations should therefore omit the Accept-encoding header entirely or if it has to be included, include "identity" or the wildcard "*" as an accepted content-encoding type.

Use of chunked encoding is given as a SHOULD NOT rather than a MUST NOT because support for it is required by [RFC2616]. Nevertheless, this form of encoding is strongly discouraged, as the data quantities being transferred (1-2kB) make it entirely unnecessary, and support for this encoding form is vulnerable to various implementation bugs, some of which may affect security. However, implementors should be aware that many versions of the Apache web server will unnecessarily use chunked encoding when returning responses. Although it would be better to make this a MUST NOT, this would render clients that rejected it incompatible with the world's most widely used web server. For this reason, support for chunked encoding is strongly discouraged but is nevertheless permitted. Clients that choose not to support it should be aware that they may run into problems when communicating with Apache-based HTTP certificate stores.

Multiple responses are returned as multipart/mixed rather than an ASN.1 SEQUENCE OF Certificate or PKCS #7/CMS certificate chain (degenerate signed data containing only certificates) because this is

more straightforward to implement with standard web-enabled tools. An additional advantage is that it doesn't restrict this access mechanism to DER-based data, allowing it to be extended to other certificate types, such as XML, PGP, and SPKI.

2.5.5. Performance Issues

Where high throughput/performance under load is a critical issue, a main-memory database that acts as a form of content cache may be interposed between the on-disk database and the HTTP interface [Garcia-Molina]. A main-memory database provides the same functionality as an on-disk database and is fully transparent to the HTTP front-end, but offers buffer management and retrieval facilities optimised for memory-resident data. Where further scalability is required, the content-caching system could be implemented as a cluster of main-memory databases [Ji].

Various network efficiency considerations need to be taken into account when implementing this certificate/public key distribution mechanism. For example, a simplistic implementation that performs two writes (the HTTP header and the certificate, written separately) followed by a read will interact badly with TCP delayed-ACK and slow-start. This occurs because the TCP MSS is typically 1460 bytes on a LAN (Ethernet) or 512/536 bytes on a WAN, while HTTP headers are ~200-300 bytes, far less than the MSS. When an HTTP message is first sent, the TCP congestion window begins at one segment, with the TCP slow-start then doubling its size for each ACK. Sending the headers separately will send one short segment and a second MSS-size segment, whereupon the TCP stack will wait for the responder's ACK before continuing. The responder gets both segments, then delays its ACK for 200ms in the hopes of piggybacking it on responder data, which is never sent, since it's still waiting for the rest of the HTTP body from the initiator. As a result, there is a 200ms (+assorted RTT) delay in each message sent.

There are various other considerations that need to be taken into account to provide maximum efficiency. These are covered in depth elsewhere [Spero] [Heidemann] [Nielsen]. In addition, modifications to TCP's behaviour, such as the use of 4K initial windows [RFC3390] (designed to reduce small HTTP transfer times to a single RTT), should also ameliorate some of these issues.

A rule of thumb for optimal performance is to combine the HTTP header and data payload into a single write (any reasonable HTTP implementation will do this anyway, thanks to the considerable body of experience that exists for HTTP server performance tuning), and to keep the HTTP headers to a minimum to try to fit data within the TCP MSS. For example, since this protocol doesn't involve a web browser,

there's no need to include various common browser-related headers such as ones detailing software versions or acceptable languages.

2.5.6. Miscellaneous

The interface specified in this document is a basic read-only type that will be used by the majority of clients. The handling of updates (both insertion and deletion) is a complex issue involving both technological issues (a variety of fields used for indexing and information retrieval need to be specified in a technology-neutral manner, or the certificate store needs to perform its own parsing of the item being added, moving it from a near-universal key=value lookup mechanism to a full public-key/certificate processing system) and political ones (who can perform updates to the certificate store, and under what conditions?). Because of this complexity, the details of any potential update mechanism are left as a local configuration issue, although they may at some point be covered in a future document if there is sufficient demand.

Concerns have been raised over the use of HTTP as a substrate [RFC3205]. The mechanism described here, which implements a straightforward request/response protocol with the same semantics as traditional HTTP requests, is unaffected by these issues. Specifically, it does not implement any form of complex RPC mechanism, does not require HTTP security measures, is not affected by firewalls (since it uses only a basic HTTP GET rather than layering a new protocol on top of HTTP), and has well-defined MIME media types specified in standards documents. As such, the concerns expressed in [RFC3205] do not apply here. In addition, although a number of servers still don't fully support some of the more advanced features of HTTP 1.1 [Krishnamurthy], the minimal subset used here is well supported by the majority of servers and HTTP implementations.

This access mechanism is similar to the PGP HKP protocol [HKP]; however, the latter is almost entirely undocumented and requires that implementors reverse-engineer other implementations. Because of this lack of standardisation, no attempt has been made to ensure interoperability or compatibility with HKP-based servers, although PGP developers provided much valuable input for this document. One benefit that HKP does bring is extensive implementation experience, which indicates that this is a very workable solution to the problem of a simple certificate/public key retrieval mechanism. HKP servers have been implemented using flat files, Berkeley DB, and various databases, such as Postgres and MySQL.

2.6. Examples

To convert the subject DN C=NZ, O=... CN=Fred Dagg into a search key:

Hash the DN, in the DER-encoded form it appears in the certificate, to obtain

```
96 4C 70 C4 1E C9 08 E5 CA 45 25 10 D6 C8 28 3A 1A C1 DF E2
```

Base-64 encode this to obtain:

```
lkxwxB7JCOXKRSUQ1sgoOhrB3+I
```

(Note the absence of trailing '=' padding.) This is the search key to use in the query URI.

To fetch all certificates useful for sending encrypted email to foo@example.com:

```
GET /search.cgi?email=foo%40example.com HTTP/1.1
```

(For simplicity, the additional Host: header required by [RFC2616] is omitted here and in the following examples.) In this case, "/search.cgi" is the abs_path portion of the query URI, and the request is submitted to the server located at the net_loc portion of the query URI. Note the encoding of the '@' symbol as per [RFC2854]. Remaining required headers, such as the "Host" header required by HTTP 1.1, have been omitted for the sake of clarity.

To fetch the CA certificate that issued the email certificate:

```
<Convert the issuer DN to a search key>  
GET /search.cgi?sHash=<search key> HTTP/1.1
```

Alternatively, if chaining is by key identifier:

```
<Extract the keyIdentifier from the authorityKeyIdentifier>  
GET /search.cgi?sKIDHash=<search key> HTTP/1.1
```

To fetch other certificates belonging to the same user as the email certificate:

```
<Convert the subject DN to a search key>  
GET /search.cgi?sHash=<search key> HTTP/1.1
```

To fetch the CRL for the certificate:

```
<Convert the issuer DN to a search key>
GET /search.cgi?iHash=<search key> HTTP/1.1
```

Note that since the differentiator is the URI base, the above two queries appear identical (since the URI base isn't shown) but are in fact distinct.

To retrieve a key using XML methods, the <KeyName> (which contains the string identifier for the key), used with the subject DN hash above, would be:

```
<KeyName KeyID="sHash">lkxwxB7JCOXKRSUQlsgoOhrB3+I</KeyName>.
```

3. Locating HTTP Certificate Stores

In order to locate servers from which certificates may be retrieved, relying parties can employ one or more of the following strategies:

- Information contained in the certificate
- Use of DNS SRV
- Use of a "well-known" location
- Manual configuration of the client software

The intent of the various options provided here is to make the certificate store access as transparent as possible, only requiring manual user configuration as a last resort.

3.1. Information in the Certificate

In order to convey a well-known point of information access to relying parties, CAs SHOULD use the SubjectInfoAccess (SIA) and AuthorityInfoAccess (AIA) extension [RFC3280] in certificates. The OID value for the accessMethod is one of:

```
id-ad-http-certs      OBJECT IDENTIFIER ::= { id-ad 6 }
id-ad-http-crls      OBJECT IDENTIFIER ::= { id-ad 7 }
```

where:

```
id-ad                  OBJECT IDENTIFIER ::= { iso(1)
                                identified-organization(3) dod(6)
                                internet(1) security(5) mechanisms(5)
                                pkix(7) 48 }
```

The corresponding accessLocation is the query URI. The use of this facility provides a CA with a convenient, standard location to indicate where further certificates may be found, for example, for certification path construction purposes. Note that it doesn't mean that the provision of certificate store access services is limited to CAs only.

3.2. Use of DNS SRV

DNS SRV is a facility for specifying the location of the server(s) for a specific protocol and domain [RFC2782]. For the certificate store interface, the DNS SRV symbolic name for the certificate store interface SHALL be "certificates". The name for the CRL store interface SHALL be "crls". The name for the PGP public key store SHALL be "pgpkeys". The name for the PGP revocation store SHALL be "pgprevolutions". Handling of additional DNS SRV facilities, such as the priority and weight fields, is as per [RFC2782].

3.2.1. Example

If a CA with the domain example.com were to make its certificates available via an HTTP certificate store interface, the server details could be obtained by a lookup on:

```
_certificates._tcp.example.com
```

and

```
_crls._tcp.example.com
```

This would return the server(s) and port(s) for the service as specified in [RFC2782].

3.3. Use of a "well-known" Location

If no other location information is available, the certificate store interface may be located at a "well-known" location constructed from the service provider's domain name. In the usual case, the URI is constructed by prepending the type of information to be retrieved ("certificates.", "crls.", "pgpkeys.", or "pgprevolutions.") to the domain name to obtain the net_loc portion of the URI, and by appending a fixed abs_path portion "search.cgi". The URI form of the "well-known" location is therefore:

```
certificates.<domain_name>/search.cgi  
crls.<domain_name>/search.cgi  
pgpkeys.<domain_name>/search.cgi  
pgprevolutions.<domain_name>/search.cgi
```


Certificate store service providers SHOULD use these URIs in preference to other alternatives. Note that the use of "search.cgi" does not imply the use of CGI scripts [RFC3875]. This would be the exception rather than the rule, since it would lead to a rather inefficient implementation; it merely provides one possible (and relatively simple to set up) implementation alternative (see the rationale for more on this).

A second case occurs when the certificate access service is being provided by web-enabled embedded devices, such as Universal Plug and Play devices [UPNP]. These devices have a single, fixed net_loc (either an IP address or a DNS name) and make services available via an HTTP interface. In this case, the URI is constructed by appending a fixed abs_path portion "certificates/search.cgi" for certificates, "crls/search.cgi" for CRLs, "pgpkeys/search.cgi" for PGP public keys, and "pgprevolutions/search.cgi" for PGP revocation information to the net_loc. The URI form of the "well-known" location is therefore:

```
<net_loc>/certificates/search.cgi
<net_loc>/crls/search.cgi
<net_loc>/pgpkeys/search.cgi
<net_loc>/pgprevolutions/search.cgi
```

If certificate access as described in this document is implemented by the device, then it SHOULD use these URIs in preference to other alternatives (see the rationale for more on this requirement).

3.3.1. Examples

If a CA with the domain example.com were to make its certificates available via an HTTP certificate store interface, the "well-known" query URIs for certificates and CRLs would be:

```
http://certificates.example.com/search.cgi
http://crls.example.com/search.cgi
```

A home automation controller with the IP address 192.0.2.1 (a control point in UPnP terminology) would make certificates for devices such as HVAC controllers, lighting and appliance controllers, and fire and physical intrusion detection devices available as:

```
http://192.0.2.1/certificates/search.cgi
http://192.0.2.1/crls/search.cgi
```

A print server with DNS name "printspooler" would make certificates for web-enabled printers that it communicates with available as:

```
http://printspooler/certificates/search.cgi
http://printspooler/crls/search.cgi
```

3.4. Manual Configuration of the Client Software

The accessLocation for the HTTP certificate/public key/CRL store MAY be configured locally at the client. This can be used if no other information is available, or if it is necessary to override other information.

3.5. Implementation Notes and Rationale

This informative section documents the rationale behind the design in Section 3 and provides guidance for implementors.

3.5.1. DNS SRV

The optimal solution for the problem of service location would be DNS SRV. Unfortunately, the operating system used by the user group most desperately in need of this type of handholding has no support for anything beyond the most basic DNS address lookups, making it impossible to use DNS SRV with anything but very recent Win2K and XP systems. To make things even more entertaining, several of the function names and some of the function parameters changed at various times during the Win2K phase of development, and the behaviour of portions of the Windows sockets API changed in undocumented ways to match. This leads to an unfortunate situation in which a Unix sysadmin can make use of DNS SRV to avoid having to deal with technical configuration issues, but a Windows'95 user can't. Because of these problems, an alternative to DNS SRV is provided for situations where it's not possible to use this.

The SRV or "well-known" location option can frequently be automatically derived by user software from currently-known parameters. For example, if the recipient's email address is @example.com, the user software would query _certificates._tcp.example.com or go to certificates.example.com and request the certificate. In addition, user software may maintain a list of known certificate sources in the way that known CA lists are maintained by web browsers. The specific mention of support for redirection in Section 2 emphasises that many sites will outsource the certificate-storage task. At worst, all that will be required is the addition of a single static web page pointing to the real server. Alternatives such as DNS CNAME RRs are also possible but may not be as easy to set up as HTTP redirects (corporate policies tend to be

more flexible in regard to web page contents than modifying DNS configurations would be).

3.5.2. "well-known" Locations

The "well-known" location URI is designed to make hosting options as flexible as possible. Locating the service at `www.<domain name>` would generally require that it be handled by the provider's main web server, while using a distinct server URI allows for it be handled as desired by the provider. Although there will no doubt be servers that implement the interface using Apache and Perl scripts, a more logical implementation would consist of a simple network interface to a key-and-value lookup mechanism, such as Berkeley DB. The URI form presented in Section 3.3 allows for maximum flexibility, since it will work with both web servers/CGI scripts and non-web-server-based network front-ends for certificate stores.

3.5.3. Information in the Certificate

Implementations that require the use of nonstandard locations, ports, or HTTPS rather than HTTP in combination with "well-known" locations should use an HTTP redirect at the "well-known" location to point to the nonstandard location. For example, if the print spooler in Section 3.3 used an SSL-protected server named `printspooler-server` with an `abs_path` portion of `cert_access`, it would use an HTTP 302 redirect to `https://printspooler-server/cert_access`. This combines the plug-and-play capability of "well-known" locations with the ability to use nonstandard locations and ports.

The SIA and AIA extensions are used to indicate the location for the CRL store interface rather than the `CRLDistributionPoint` (CRLDP) extension, since the two perform entirely different functions. A CRLDP contains "a pointer to the current CRL", a fixed location containing a CRL for the current certificate, while the SIA/AIA extension indicates "how to access CA information and services for the subject/issuer of the certificate in which the extension appears", in this case, the CRL store interface that provides CRLs for any certificates issued by the CA. In addition, CRLDP associates other attribute information with a query that is incompatible with the simple query mechanisms presented in this document.

A single server can be used to handle both CRLDP and AIA/SIA queries provided that the CRLDP form uses an HTTP URI. Since CRLDP points to a single static location for a CRL, a query can be pre-constructed and stored in the CRLDP extension. Software that uses the CRLDP will retrieve the single CRL that applies to the certificate from the server, and software that uses the AIA/SIA can retrieve any CRL from the server. Similar pre-constructed URIs may also be useful in other

circumstances (for example, for links on web pages) to place in appropriate locations like the issuerAltName, or even for technical support/helpdesk staff to email to users who can't find the certificate themselves, as described in Section 2.5. The resulting certstore URL, when clicked on by the user, will directly access the certificate when used in conjunction with any certificate-aware application, such as a browser or mail program.

3.5.4. Miscellaneous

Web-enabled (or, more strictly, HTTP-enabled) devices are intended to be plug-and-play, with minimal (or no) user configuration necessary. The "well-known" URI allows any known device (for example, one discovered via UPnP's Simple Service Discovery Protocol, SSDP) to be queried for certificates without requiring further user configuration. Note that in practice no embedded device would ever use the address given in the example (the de facto standard address for web-enabled embedded devices is 192.168.1.x and not 192.0.2.x); however, IETF policy requires the use of this non-address for examples.

Protocols such as UPnP have their own means of disseminating device and protocol information. For example, UPnP uses SOAP, which provides a GetPublicKeys action for pulling device keys and a PresentKeys action for pushing control point keys. The text in Section 3.3 is not meant to imply that this document overrides the existing UPnP mechanism, but merely that, if a device implements the mechanism described here, it should use the naming scheme in Section 3.3 rather than use arbitrary names.

4. Security Considerations

HTTP caching proxies are common on the Internet, and some proxies may not check for the latest version of an object correctly. [RFC2616] specifies that responses to query URLs should not be cached, and most proxies and servers correctly implement the "Cache-Control: no-cache" mechanism, which can be used to override caching ("Pragma: no-cache" for HTTP 1.0). However, in the rare instance in which an HTTP request for a certificate or CRL goes through a misconfigured or otherwise broken proxy, the proxy may return an out-of-date response.

Care should be taken to ensure that only valid queries are fed through to the back-end used to retrieve certificates. Allowing attackers to submit arbitrary queries may allow them to manipulate the certificate store in unexpected ways if the back-end tries to interpret the query contents. For example, if a certificate store is implemented using an RDBMS for which the calling application assembles a complete SQL string to perform the query, and the SQL

query is built up as "SELECT certificate FROM certificates WHERE iHash = " + <search key>, and <search key> is set to "X;DELETE FROM certificates", the results of the query will be quite different from what was expected by the certificate store administrator. The same applies to queries by name and email address. Even a read-only query can be problematic; for example, setting <search key> to "UNION SELECT password FROM master.sysxlogins" will list all passwords in an SQL Server database (in an easily decrypted format) if the user is running under the sa (DBA) account. Straightforward sanitisation of queries may not be sufficient to prevent all attacks; for example, a filter that removes the SQL query string "DELETE" can be bypassed by submitting the string embedded in another instance of the string. Removing "DELETE" from "DELDELETEETE" leaves the outer "DELETE" in place. Abusing the truncation of over-long strings by filters can also be used as a means of attack, in which the attacker ensures that the truncation occurs in the middle of an escape sequence, bypassing the filtering. The use of parameterised queries (often called placeholders) that aren't vulnerable to SQL injection should be used to avoid these attacks.

In addition, since some query data may be encoded/decoded before being sent to the back-end, applications should check both the encoded and decoded form for valid data. A simple means of avoiding these problems is to use parameterised commands rather than hand-assembling SQL strings for use in queries (this is also more efficient for most database interfaces). The use of parameterised commands means that the query value is never present in any position where it could be interpreted as a portion of the query command.

Alongside filtering of queries, the back-end should be configured to disable any form of update access via the web interface. For Berkeley DB, this restriction can be imposed by opening the certificate store in read-only mode from the web interface. For relational databases, it can be imposed through the SQL GRANT/REVOKE mechanism, for example, "REVOKE ALL ON certificates FROM webuser. GRANT SELECT ON certificates TO webuser" will allow read-only access of the appropriate kind for the web interface. Server-specific security measures may also be employed; for example, the SQL Server provides the built-in db_datareader account that only allows read access to tables (but see the note above about what can be done even with read-only access) and the ability to run the server under a dedicated low-privilege account (a standard feature of Unix systems).

The mechanism described in this document is not intended to function as a trusted directory/database. In particular, users should not assume that just because they fetched a public key or certificate from an entity claiming to be X, that X has made any statement about the veracity of the public key or certificate. The use of a signed

representation of the items stored removes the need to depend on the certificate store for any security service other than availability. Although it's possible to implement a trusted directory/database using HTTPS or some other form of secured/trusted link, this is a local policy/configuration issue, and in the absence of such additional security measures users should apply appropriate levels of verification to any keys or certificates fetched before they take them into use.

5. IANA Considerations

No action by IANA is needed. The AIA/SIA accessMethod types are identified by object identifiers (OIDs) from an arc managed by the PKIX working group. Should additional accessMethods be introduced (for example, for attribute certificates or non-X.509 certificate types), the advocates for such accessMethods are expected to assign the necessary OIDs from their own arcs.

6. Acknowledgements

Anders Rundgren, Blake Ramsdell, Jeff Jacoby, David Shaw, and members of the ietf-pkix working group provided useful input and feedback on this document.

7. References

7.1. Normative References

- [FIPS180] Federal Information Processing Standards Publication (FIPS PUB) 180-1, Secure Hash Standard, 17 April 1995.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2440] Callas, J., Donnerhackle, L., Finney, H., and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998.
- [RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", RFC 2585, May 1999.
- [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.

- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC2854] Connolly, D. and L. Masinter, "The 'text/html' Media Type", RFC 2854, June 2000.
- [RFC3156] Elkins, M., Del Torto, D., Levien, R., and T. Roessler, "MIME Security with OpenPGP", RFC 3156, August 2001.
- [RFC3275] Eastlake 3rd, D., Reagle, J., and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", RFC 3275, March 2002.
- [RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.
- [RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

7.2. Informative References

- [Birkholz] "Special Ops: Host and Network Security for Microsoft, Unix, and Oracle", Erik Birkholz et al, Syngress Publishing, November 2002.
- [Garcia-Molina] "Main Memory Database Systems", Hector Garcia-Molina and Kenneth Salem, IEEE Transactions on Knowledge and Data Engineering, Vol.4, No.6 (December 1992), p.509.
- [Gutmann] "A Reliable, Scalable General-purpose Certificate Store", P. Gutmann, Proceedings of the 16th Annual Computer Security Applications Conference, December 2000.
- [Heidemann] "Performance Interactions Between P-HTTP and TCP Implementations", J. Heidemann, ACM Computer Communications Review, April 1997.
- [HKP] "A PGP Public Key Server", Marc Horowitz, 2000, <http://www.mit.edu/afs/net.mit.edu/project/pks/thesis/paper/thesis.html>. A more complete and up-to-date overview of HKP may be obtained from the source code of an open-source OpenPGP implementation such as GPG.

- [Ji] "Affinity-based Management of Main Memory Database Clusters", Minwen Ji, ACM Transactions on Internet Technology, Vol.2, No.4 (November 2002), p.307.
- [Krishnamurthy] "PRO-COW: Protocol Compliance on the Web - A Longitudinal Survey", Balachander Krishnamurthy and Martin Arlitt, Proceedings of the 3rd Usenix Symposium on Internet Technologies and Systems (USITS'01), March 2001, p.109.
- [Nielsen] "Network Performance Effects of HTTP/1.1, CSS1, and PNG", H.Nielsen, J.Gettys, A.Baird-Smith, E.Prud'hommeaux, H.Wium Lie, and C.Lilley, 24 June 1997, <http://www.w3.org/Protocols/HTTP/Performance/Pipeline.html>
- [PKCS11] PKCS #11 Cryptographic Token Interface Standard, RSA Laboratories, December 1999.
- [PKCS15] PKCS #15 Cryptographic Token Information Syntax Standard, RSA Laboratories, June 2000.
- [RFC3205] Moore, K., "On the use of HTTP as a Substrate", BCP 56, RFC 3205, February 2002.
- [RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.
- [RFC3875] Robinson, D. and K. Coar, "The Common Gateway Interface (CGI) Version 1.1", RFC 3875, October 2004.
- [Spero] "Analysis of HTTP Performance Problems", S.Spero, July 1994, <http://www.w3.org/Protocols/HTTP/1.0/HTTPPerformance.html>.
- [UPNP] "Universal Plug and Play Device Architecture, Version 1.0", UPnP Forum, 8 June 2000.

Author's Address

Peter Gutmann
University of Auckland
Private Bag 92019
Auckland, New Zealand

EMail: pgut001@cs.auckland.ac.nz

Full Copyright Statement

Copyright (C) The Internet Society (2006).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

