Internet Draft
draft-ietf-pkix-certstore-http-04.txt
February 2003
Expires August 2003

Editor: Peter Gutmann
University of Auckland

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

Status of this memo

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.

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 (throughout the remainder of this document the generic term HTTP will be used to cover either option) as an interface mechanism to obtain certificates and certificate revocation lists (CRLs) from PKI repositories.

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 allows HTTP to be used as a general-purpose, transparent interface to any type of certificate store ranging from flat files through to standard databases such as Berkeley DB and relational databases, as well as traditional X.500/LDAP directories. Typical applications would include use with web-enabled relational databases (which most current 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].

This draft is being discussed on the "ietf-pkix" mailing list. To join the list, send a message to <ietf-pkix-request@imc.org> with the single word "subscribe" in the body of the message. Also, there is a Web site for the mailing list at <http://www.imc.org/ietf-pkix>.

2. HTTP Certificate Store Interface

The GET method is used in combination with a query URI to retrieve certificates from the underlying certificate store [RFC2068]. The parameters for the query URI are a certificate identifier consisting of an attribute type and a value that specifies one or more certificates to be returned from the query. The query URI may be specified in a certificate SubjectInfoaccess or AuthorityInfoAccess extension or configured at the client (see section 3).

Permitted attribute types and associated values are described below. Arbitrary-length binary values (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, and are therefore case-sensitive.

Attribute	Binary	Value
certHash	Y	Search key derived from the SHA-1 hash of the certificate (sometimes called the certificate fingerprint or thumbprint).
email	N	Subject email address associated with the certificate.
iHash	Y	Search key derived from the issuer DN as it appears in the certificate, CRL, or other object.
iAndSHash	Y	Search key derived from the certificate's issuerAndSerialNumber [RFC2630].
name	N	Subject CommonName contained in the certificate.
sHash	Y	Search key derived from the subject DN as it appears in the certificate or other object.
sKIDHash	Y	Search key derived from the certificate's

subjectKeyIdentifier.

The full URI is formed by concatenating the query URI and the attribute and value. Certificates are retrieved from one query URI (the certificate URI) and CRLs from another query URI (the CRL URI). These may or may not correspond to the same certificate store and/or server (the exact interpretation is a local configuration issue). The form of the complete URI is therefore:

<query URI> '?' <attribute> '=' <value>

The query value MUST be encoded using the form-urlencoded media type [RFC1866]. Further details of URI construction, size limits, and other details are given in [RFC2068].

Certificate URIs MUST support retrieval by all of the above attribute types. CRL URIs MUST support retrival by the iHash and sKID attribute types, which identify the issuer of the CRL. A CRL query MUST return the matching CRL with the greatest thisUpdate value (in other words, the most recent CRL).

If more than one certificate matches a query, it MUST be returned as a multipart/mixed response.

Responses to unsuccessful queries (for example to indicate a non-match or an error conditions) are handled in the standard manner as per [RFC2068]. 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.

Other information such as naming conventions and MIME types are specified in [RFC2585].

2.1 Converting Binary Blobs into Search Keys

The fields marked as binary data in the table in section 2 are of arbitrary length and contain non-textual data. Both of these properties make them unsuited for direct use in HTTP queries. In order to make them usable, they are first hashed down to a fixed-length 160-bit value and then base64-encoded:

Step 1: Hash the key value using SHA-1 to produce a 160-bit value.

Step 2: Encode the hash value using base64 encoding to produce a
        27-byte text-only value, excluding any trailing '=' padding
        values that are sometimes used with base64 encoding.

Certificate stores MUST verify that the base64-encoded values submitted in requests contain only characters in the range 'a'-'z', 'A'-'Z', '0'-'9', '+', and '/'. Queries containing any other character MUST be rejected (see the implementation notes in section 2.2 and the security considerations in section 4 **for more details on this requirement).**

2.2 Implementation Notes

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

The base64-encoded form of the identifier should be carefully checked for invalid characters since allowing raw data through presents a security risk. Consider for example a certificate 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. For this reason only valid base64 encodings should be allowed. The same checking applies to queries by name or email address.

Pre-constructed URIs that fetch a certificate matching a fixed search criterion may be useful for situations such as web pages or business cards, or even for technical support/helpdesk staff to mail to users who 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 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. Simiarly, 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).

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), has well-defined MIME media types specified in standards documents, etc etc etc. As such, the concerns expressed in [RFC3205] do not apply here.

Various network efficiency considerations need to be taken into account when implementing this certificate distribution mechanism. For example, a simplistic implementation that performs two writes (the HTTP header and the certificate written seperately) 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, this results in a 200ms (+ assorted RTT) delay in each message sent.

There are various other considerations that need to be taken into account in order 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 and fit data within the TCP MSS. Since this protocol doesn't involve a web browser, there's no need to include the usual headers covering browser versions and languages and so on; a minimal set of content-type/encoding and host and session control information will suffice.

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.

2.3 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@bar.com:

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

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 [RFC1866]. 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?iHash=<search key> HTTP/1.1
```

Alternatively, if chaining is by key identifier:

```
<Extract the keyIdentifier from the authorityKeyIdentifier>
GET /search-cgi?sKID=<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.

2.4 Rationale

The identifiers are taken from PKCS #15 [PKCS15], a standard that covers (among other things) a transparent interface to a certificate store. These identifiers have been field proven through having 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].

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 query types have been specifically chosen to be not just an HTTP interface to LDAP but as a general-purpose retrieval mechanism that allows arbitrary certificate 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 the use of 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.

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 the fact 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.

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, email, etc etc) where multiple name forms exist.

Multiple response are returned as multipart/mixed rather than an ASN.1 SEQUENCE OF Certificate or PKCS #7/CMS certificate chain 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.

Certificate 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 independant URIs it's possible to implement the two stores using mechanisms tailored to the data they contain.

This access mechanism is similar to the PGP HKP protocol, 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. 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 key/certificate retrieval mechanism. HKP servers have been implemented using flat files, Berkeley DB, and various databases such as Postgres and MySQL.

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 to relying parties a well-known point of information access, CAs MAY use of 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 }
```

and the corresponding accessLocation is the query URI.

This provides a CA with a convenient place to indicate where further certificates may be found, for example for 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".

3.2.1 Example

If a CA with the domain kiwisign.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.kiwisign.com
```

and

```
  _crls._tcp.kiwisign.com
```

which 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, either "certificates." or "crls.", to the domain name to obtain the net_loc portion of the URI and 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
```

Service providers SHOULD use these URIs in preference to other alternatives.

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 and "crls/search.cgi" for CRLs to the net_loc. The URI form of the "well-known" location is therefore:

```
<net_loc>/certificates/search.cgi
<net_loc>/crls/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 kiwisign.com were to make its certificates available via an HTTP certificate store interface, the "well-known" query URIs for certificates and CRLs would be:

```
certificates.kiwisign.com/search.cgi
crls.kiwisign.com/search.cgi
```

A home automation controller with IP address 192.168.1.1 (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:

```
192.168.1.1/certificates/search.cgi
192.168.1.1/crls/search.cgi
```

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

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

3.4 Manual Configuration of the Client Software

The accessLocation for the HTTP certificate/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

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 @hotmail.com, the user software would query _certificates._tcp.hotmail.com or go to certificates.hotmail.com and request the certificate. If the recipient worked for a government department, the certificate would be requested via _certificates._tcp.departmentname.gov or at certificates.departmentname.gov. 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 the fact 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 obviously also possible, but aren't quite as easy to set up as HTTP redirects and won't work well across domains.

Implementations that require the use of nonstandard locations or 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.

A single server can be used to handle both CRLDP and AIA/SIA queries provided 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 2.2.

3.6 Rationale

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.

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 the 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 well-known location URI is designed to make hosting options as flexible as possible. Locating the service at www.<domain name> would generally require it to be handled by the provider's main web server, while using a distinct server URI allows it to 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.

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.

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 describe here, it should use the naming scheme in section 3.3 rather than using 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. [RFC2068] specifies that responses to query URLs should not be cached, and most proxies and servers correctly implement the "Cache-Control: no-cache" mechanism that can be used to override cacheing ("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 backend used to retrieve certificates. Allowing an attacker to submit arbitrary queries may allow them to manipulate the certificate store in unexpected ways if the backend tries to interpret the query contents. For example if a certificate store is implemented using an RDBMS in which 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.

Alongside filtering of queries, the backend 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.

4. IANA Considerations

The AIA/SIA accessMethod types are identified by object identifiers (OIDs). OIDs were assigned from an arc contributed to the PKIX Working Group by RSA Security. 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. No action by the IANA is necessary for this document or any anticipated updates.

Acknowledgements

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

Author Address

Peter Gutmann
University of Auckland
Private Bag 92019
Auckland, New Zealand
pgut001@cs.auckland.ac.nz

References (Normative)

[RFC1866] "Hypertext Markup Language - 2.0", RFC 1866, T. Berners-Lee and D. Connolly, November 1995.

[RFC2068] "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, January 1997.

[RFC2119] "Key Words for Use in RFCs to Indicate Requirement Levels", RFC 2119, S.Bradner, March 1997.

[RFC3280] "Internet X.509 Public Key Infrastructure: Certificate and CRL Profile", RFC 3280, R. Housley, W. Ford, W. Polk, and D. Solo, April 2002.

[RFC2585] "Internet X.509 Public Key Infrastructure: Operational Protocols: FTP and HTTP", RFC 2585, R. Housley and P. Hoffman, May 1999

[RFC2782] "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, A.Gulbrandsen, P.Vixie, and L.Esibov, February 2000.

References (Informative)

[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.

[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/1.0/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] "On the use of HTTP as a substrate", RFC 3205, K.Moore, February 2002.

[RFC3390] "Increasing TCP's Initial Window", RFC 3390, M.Allman, S.Floyd, and C.Partridge, October 2002.

[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.

Full Copyright Statement