Internet-Draft
Expires September 2003

D. Shaw
Jabberwocky Tech
March 2003

The OpenPGP HTTP Keyserver Protocol (HKP)
draft-shaw-openpgp-hkp-00.txt

Status of this Memo

Abstract

This document specifies a series of conventions to implement an OpenPGP keyserver using the Hypertext Transfer Protocol (HTTP). As this document is a codification and extension of a protocol that is already in wide use, strict attention is paid to backward compatibility with these existing implementations.

Table of Contents

1. Introduction

For ease of use, public key cryptography requires a key distribution system. For many years, the most commonly used system has been a key server - a server that stores public keys and can be searched for a given key. The HTTP Keyserver Protocol is a OpenPGP keyserver implemented using HTTP.

1.1. Terms

This document uses the terms "MUST", "SHOULD" and "MAY" as defined in RFC-2119 [1], along with the negated forms of those terms.

2. HKP and HTTP

As HKP is implemented over HTTP, everything in RFC-1945 [2] applies to HKP as well, and HKP error codes are the same as the ones used in HTTP. In order to give as much information to the client about error conditions, is good practice to return the most specific error code possible: for example, returning 404 ("Not Found") rather than 400 ("Bad Request") when a key is not found.

This document gives suggested HTTP error codes for several common situations. Note that these are only suggestions, and implementations may have good reasons (such as not revealing the reason why a request failed) for using other error codes.

Due the very large deployment of HKP clients based on HTTP version 1.0, HKP keyservers MUST support HTTP 1.0. HKP keyservers MAY additionally support other HTTP versions.

[dshaw : I expect this to be controversial, but we've got tons of deployed code that only works with 1.0. I'd be willing to discuss

removing this MUST or make it a SHOULD and add a "implementation notes" section pointing out the problem instead.]

By convention and history, HKP uses HTTP on TCP port 11371. It has

been suggested by some that for reasons of maximum compatibility with firewalls and filtering HTTP proxies, it is better to use the standard HTTP port (TCP port 80). See section 7, Locating a HKP Keyserver for an automated way for clients to discover the correct port.

3. Requesting Data From A Keyserver

Keyserver requests are done via a HTTP GET URL that encodes the request within it. Specifically, the abs_path (see [2], section 3.2) is built up of the base request "/pks/lookup", followed by any variables. Arguments are passed through the usual means as specified in [3], section 8.2.2. The variables may be given in any order. Keyservers MUST ignore any unknown variables.

3.1. Basic Variables

All HKP requests contain the "op" (operation) and "search" variables. The "op" variable determines what operation the keyserver will take, and the "search" variable determines that keys are operated on.

3.1.1. The "search" Variable

The search variable contains arbitrary text encoded as usual for a HTTP URL. This text may represent the key ID of the key being sought or some text from a user ID on the key being sought.

If any particular type of searching is not supported, the keyserver should return an appropriate HTTP error code such as 501 ("Not Implemented"). The server MUST NOT return an error code (such as 404 ("Not Found")) that could be mistaken by the client for a valid response.

3.1.1.1. Key ID and V4 Fingerprint Searches

If a key is being sought by its key ID, the key ID string is prefixed with an "0x" to indicate a hexadecimal number. Key ID strings may be 8 digits (32-bit key ID), 16 digits (64-bit key ID), 32 digits (version 3 fingerprint), or 40 digits (version 4 fingerprint). The hexadecimal digits are not case sensitive.

A keyserver that allows searching by keyid MUST accept the 160-bit version 4 fingerprint, 64-bit key IDs, and 32-bit key IDs in the "search" variable. Note this does not mean that the keyserver will actually use the full length of the request in the search, as it may internally create a 32-bit or 64-bit key ID from a version 4 fingerprint (by taking the low-order 32 or 64 bits respectively), or a 32-bit key ID from a 64-bit key ID (by taking low-order 32 bits). That said, a keyserver SHOULD use at least 64 bits of the

key ID if available.

3.1.1.2. V3 Fingerprint Searches

The 128-bit version 3 fingerprint is represented by a leading "0x", followed by 32 case-insensitive hexadecimal digits. Note that v3 fingerprints are treated differently and not grouped with keyid or v4 fingerprint searches as it is not possible to calculate a 64-bit or 32-bit keyid from a v3 fingerprint.

3.1.1.3. Text Searches

How a keyserver handles a textual search is implementation defined. See also the definition of the "exact" variable for a method to give additional instructions to the server on how the search is to be executed.

3.1.2. The "op" (operation) Variable

The op variable specifies the operation to be performed on the keyserver. The op variable is generally used with the "search" variable to specify the keys that should be operated on.

3.1.2.1. The "get" operation

The "get" operation requests keys from the keyserver. A string that specifies which key(s) to return is provided in the "search" variable.

The response to a successful "get" request is a HTTP document containing a keyring as specified in RFC-2440 [4], section 11.1, and ASCII armored as specified in section 6.2.

The response may be wrapped in any HTML or other text desired, except that the actual key data consisting of an initial line break, the "-----BEGIN PGP PUBLIC KEY BLOCK-----" header, the armored key data itself, the "-----END PGP PUBLIC KEY BLOCK-----" header, and a final line break MUST NOT be modified from the form specified in [4].

If no keys match the request, the keyserver should return an appropriate HTTP error code such as 404 ("Not Found").

3.1.2.2. The "index" Operation

The "index" operation requests a list of keys on the keyserver that match the text or key ID in the "search" variable. Historically, the "index" operation returned a human readable HTML document containing links for each found key, but this is not required.

If the "index" operation is not supported, the keyserver should return an appropriate HTTP error code such as 501 ("Not Implemented").

3.1.2.3. The "vindex" (verbose index) Operation

The "vindex" operation is similar to "index" in that it provides a list of keys on the keyserver that match the text of key ID in the "search" variable. Historically, a "vindex" response was the same as

"index" with the addition of showing the signatures on each key, but this is not required.

If the "vindex" operation is not supported, the keyserver should return an appropriate HTTP error code such as 501 ("Not Implemented").

3.1.2.4. Other Operations

Other site-specific or nonstandard operations can be indicated by prefixing the operation name with the string "x-".

3.2. Modifier Variables

These variables are used to modify basic requests.

3.2.1. The "options" Variable

This variable takes one or more arguments, separated by commas. These are used to modify the behavior of the keyserver on a per-request basis.

3.2.1.1. The "mr" (Machine Readable) Option

The machine readable option instructs the server that a program (rather than a person) is making the request, so the output may be customized for that use. See Section 5, Output Formats for the specific details of machine readable output.

3.2.1.2. The "nm" (No Modification) Option

As keyservers may modify submitted keys to suit a particular policy, this option is used to inform the keyserver that the submitter would rather have the submission fail completely then have the submitted key(s) modified. An example of this would be a keyserver that does not accept user IDs with an email address outside of the local domain. If such a key was submitted, the keyserver could trim any noncompliant user IDs before accepting the key. If this option was set, then the key submission would fail.

3.2.1.3. Other Options

Other site-specific or nonstandard options can be indicated by prefixing the option name with the string "x-".

3.2.2. The "fingerprint" Variable

This variable takes one argument: "on" or "off". If present and on, it instructs the server to provide the key fingerprint for each key in an "index" or "vindex" operation. This variable has no

effect on any other operation. The exact format of the displayed fingerprint, like the "index" and "vindex" operations themselves, is implementation defined.

3.2.3. The "exact" Variable

This variable takes one argument: "on" or "off". If present and on, it instructs the server to search for an exact match for the contents of the "search" variable. The exact meaning of "exact match" is implementation defined.

3.2.3. Other Variables

Other site-specific or nonstandard variables can be indicated by prefixing the variable name with the string "x-".

3.4. Request Examples

Search for all keys containing the string "dshaw":
http://keys.example.com:11371/pks/lookup?search=dshaw&op=index

Get key 0x99242560 (32-bit key ID):
http://keys.example.com:11371/pks/lookup?op=get&search=0x99242560

4. Submitting Keys To A Keyserver

Keyserver submissions are done via a HTTP POST URL. Specifically, the abs_path (see [2], section 3.2) is set to "/pks/add", and the key data is provided via HTTP POST as specified in [2], section 8.3, and [3], section 8.2.3.

The body of the POST message contains a "keytext" variable which is set to an ASCII armored keyring as specified in [4], sections 6.2 and 11.1. The ASCII armored keyring should also be urlencoded as specified in [3], section 8.2.1. Note that more than one key may be submitted in a single transaction.

There may also be an "options" variable, as specified in section 3.2.1 above.

If a keyserver does not support adding keys via HTTP, then requests to do so should return an appropriate HTTP error code, such as 403 ("Forbidden") if key submission has been disallowed, or 501 ("Not Implemented") if the server does not support HTTP key submission. The keyserver MUST NOT return an error code (such as 404 ("Not Found")) that could be mistaken by the client for a valid response.

5. Output Formats

HKP is intended for both human and programmatic use. The "machine readable" option is used to tailor the output for a given use. In general, the "human readable" output is implementation specific. For interoperability, the "machine readable" output MUST carefully follow the guidelines given here.

Note that there is an installed base of programs that do in fact attempt to parse the human readable "index" format used in the pksd keyserver. Care should be taken with the choice of an "index" format

if compatibility with these programs is desired.

5.1. Machine Readable Output

When machine readable output is requested, several changes are made to output:

- Key retrievals (op=get) do not contain any text aside from the ASCII armored keyring. The document is also sent to the user using Content-Type: application/pgp-keys as specified in RFC-3156 [6].

- Key indexes (op=index) use the format specified in Section 5.2, Machine Readable Indexes.

5.2. Machine Readable Indexes

The machine readable index format is a list of records that can be easily parsed by a machine. The document is 7-bit clean, and as such is sent with no encoding and Content-Type: text/plain.

The machine readable response begins with an optional information line:

info:<version>:<count>

<version> = this is the version of this output format.
Currently, this is the number 1.

<count> = the number of keys returned in this response. Note this is the number of keys, and not the number of lines returned. That is, it should match the number of "pub:" lines returned.

If this optional line is not included, or the version information is not supplied, the version number is assumed to be 1.

The key listings themselves are made up of several lines per key. The first line specifies the primary key:

pub:<keyid>:<algo>:<keylen>:<creationdate>:<expirationdate>:<flags>

<keyid> = this is either the fingerprint or the key ID of the key. Either the 16-digit or 8-digit key IDs are acceptable, but obviously the fingerprint is best. A keyserver should use the most specific of the key IDs that it has available. Since it is not possible to calculate the key ID from a V3 key fingerprint, for V3 keys this should be either the 16-digit or 8-digit key ID only.

<algo> = the algorithm number from [4]. (i.e. 1==RSA, 17==DSA, etc).

<keylen> = the key length (i.e. 1024, 2048, 4096, etc.)

<creationdate> = creation date of the key in standard RFC-2440 [4] form (i.e. number of seconds since 1/1/1970 UTC time)

<expirationdate> = expiration date of the key in standard RFC-2440 [4] form (i.e. number of seconds since 1/1/1970 UTC time)

<flags> = letter codes to indicate details of the key, if any. Flags may be in any order. The meaning of "disabled" is implementation-specific. Note that individual flags may be unimplemented, so the absence of a given flag does not necessarily mean the absence of the detail.

r == revoked
d == disabled
e == expired

Following the "pub" line are one or more "uid" lines to indicate user IDs on the key:

uid:<escaped uid string>:<creationdate>:<expirationdate>:<flags>

<escaped uid string> = the user ID string, with HTTP %-escaping for anything that isn't 7-bit safe as well as for the ":" character. Any other characters may be escaped, as desired.

creationdate, expirationdate, and flags mean the same here as in the "pub" line. The information is taken from the self-signature, if any, and applies to the user ID in question, and not to the key as a whole.

Note that empty fields are allowed. For example, a key with no expiration date would have the <expirationdate> field empty. Also, a keyserver that does not track a particular piece of information may leave that field empty as well. Colons for empty fields on the end of each line may be left off, if desired.

6. Extended Status Codes

As HKP is implemented over HTTP, when a status or error code needs to be returned, the most appropriate HTTP code should be used.

Occasionally there is a need to express a condition that cannot be expressed via the HTTP 1.0 status code list. In these cases, an additional HTTP header may be added to the response. This

additional header is of the form "X-HKP-Status:" and is followed by one of the following status codes:

xxx - Submitted key was altered to match keyserver policy.

xxx - Submitted key was rejected as per keyserver policy.
xxx - The search resulted in more responses then the keyserver was willing to return.

7. Locating a HKP Keyserver

Clients are usually manually configured with the address of a HKP keyserver. Client implementors should be aware that it is reasonably common practice to use a single name in DNS that resolves to multiple address records. When receiving a DNS response with multiple addresses, clients SHOULD try each address until a server is reached. The order to try these addresses in is implementation defined.

A far more flexible scheme for listing multiple HKP keyservers in DNS is the use of DNS SRV records as specified in RFC-2782 [5]. DNS SRV allows for different priorities and weights to be applied to each HKP keyserver in the list, which allows an administrator much more control over how clients will contact the servers. The SRV symbolic service name for HKP keyservers is "hkp". For example, the SRV record for HKP keyservers in domain "example.com" would be "_hkp._tcp.example.com".

SRV records contain the port that the target server runs on, so SRV can also be used to automatically discover the proper port for contacting a HKP keyserver.

An additional use of SRV records is when a client needs to locate a specified key by email address. For example, a client trying to locate a key for isabella@silvie.example.com could consult "_hkp._tcp.silvie.example.com".

HKP clients SHOULD support SRV records.

8. Security Considerations

As described here, a keyserver is a searchable database of public keys accessed over the network. While there may be security considerations arising from distributing keys in this manner, this does not impact the security of OpenPGP itself.

Without some sort of trust relationship between the client and server, information returned from a keyserver in search results cannot be trusted by the client until the OpenPGP client actually retrieves and checks the key for itself. This is important and must be stressed: without a specific reason to treat information otherwise, all search results must be regarded as untrustworthy and informational only.

9. IANA Considerations

This document assigns the DNS SRV symbolic name "hkp".

10. Normative References

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

[2] T. Berners-Lee, "Hypertext Transfer Protocol 1.0", RFC 1945, May 1996.

[3] T. Berners-Lee, "Hypertext Markup Language - 2.0", RFC 1866, November 1995.

[4] J. Callas, L. Donnerhacke, H. Finney and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998.

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

[6] M. Elkins, D. Del Torto, R. Levien, and T. Roessler, "MIME Security with OpenPGP", RFC 3156, August 2001.

11. Author's Address

David Shaw
16 Farina Road
Newton, MA 02459

Email: dshaw@jabberwocky.com
Tel: +1 (617) 332-8443

This document is a formalization and extension of the HKP originally implemented in the PKS keyserver by Marc Horowitz, which in turn was based on earlier work by Brian LaMacchia and Michael Graff. Without their grounding, this document would not exist.

The author would also like to thank Peter Gutmann for his work on the Certstore protocol, some of which was applicable here, and the members of the pgp-keyserver-folk mailing list who contributed valuable comments and suggestions.