Workgroup: openpgp
Internet-Draft: draft-gallagher-openpgp-hkp-03
Published: 30 December 2023
Intended Status: Informational
Expires: 2 July 2024
Authors: D. Shaw
Jabberwocky Tech
A. Gallagher, Ed.
PGPKeys.EU

OpenPGP HTTP Keyserver Protocol

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.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://andrewgdotcom.gitlab.io/draft-gallagher-openpgp-hkp. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-gallagher-openpgp-hkp/.

Discussion of this document takes place on the OpenPGP Working Group mailing list (mailto:openpgp@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/openpgp/. Subscribe at https://www.ietf.org/mailman/listinfo/openpgp/.

Source for this draft and an issue tracker can be found at https://gitlab.com/andrewgdotcom/draft-gallagher-openpgp-hkp.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 2 July 2024.

Copyright Notice

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.

2. Conventions and Definitions

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

3. HKP and HTTP

As HKP is implemented over HTTP, everything in [RFC1945] applies to HKP as well, and HKP error codes are the same as the ones used in HTTP.

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. See issue #5.))

When used over HTTPS, HKP is commonly referred to as "HKPS".

By convention and history, HKP uses HTTP on TCP port 11371, and HTTPS on TCP port 443. These are often distinguished from generic use of HTTP(S) by using the URI schemes "hkp" and "hkps". For reasons of maximum compatibility with firewalls and filtering HTTP

proxies, HKP is also often served over the standard HTTP port (TCP port 80).

((andrewg : We may wish to officially recommend HKPS. See issue #7.))

See Section 8 for an automated way for clients to discover the correct port.

3.1. Request Paths

HKP defines two paths, namely "/pks/lookup" for lookups (see Section 4) and "/pks/add" for submission (see Section 5). A keyserver **MAY** support requests to other paths, but these are implementation dependent. These alternative paths have historically been used to provide human-readable interfaces such as HTML forms, and functionality extensions such as [SKS].

((andrewg : SKS uses "/pks/hashquery" for bulk updates, and hockeypuck also implements "/pks/delete" and "/pks/replace". See issue #15.))

3.2. HTTP Status Codes

When a status or error code needs to be returned by a keyserver, the most appropriate HTTP code from [RFC9110] should be used. It 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.

Clients **SHOULD** understand the following codes:

Status Code	Description
200 OK	Request succeeded
202 Accepted	Submitted key was altered to match keyserver policy
403 Forbidden	The requested operation is not permitted
404 Not found	The search returned no results, or path not found
410 Gone	Key has been permanently deleted, e.g. due to blacklisting
413 Content too large	The search returned too many responses

Status Code	Description
422 Unprocessable content	Submitted key was rejected as per keyserver policy
501 Not implemented	The requested operation is not supported

Table 1: Status Codes

In addition, a client **SHOULD** understand 3xx redirect codes.

((andrewg : In draft-shaw-00 it was suggested that a novel header be used for statuses that could not be represented by the HTTP response codes of the time. This was only partially specified, and it is unclear if any implementations of this header existed. In the meantime many new HTTP response codes have been defined, so I am using them instead - even if their semantics does not exactly match that of [RFC9110]. NB therefore that codes 202, 410, 413, 422 may not have been implemented anywhere yet. See issue #5.))

4. Looking up Data from a Keyserver

Key lookups are done via an HTTP GET request. Specifically, the abs_path (see [RFC1945], section 3.2) is built up of the base path "/pks/lookup", followed by any variables. Variables are passed using HTTP query strings as specified in [RFC1866], section 8.2.2.

Most HKP lookups contain both the "op" (operation) and "search" variables. The "op" variable determines what operation the keyserver will take, and the "search" variable determines which keys are operated on. There may also be modifier variables, as specified in Section 6 below. The variables may be given in any order. Keyservers **MUST** ignore any unknown variables.

4.1. 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.

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

4.1.1. The "get" operation

A keyserver **SHOULD** support 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 OpenPGP [RFC4880], 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 [RFC4880].

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

4.1.2. The "index" operation

A keyserver **MAY** support 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.

4.1.3. The "vindex" (verbose index) operation

A keyserver **MAY** support the "vindex" 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.

4.1.4. The "stats" (statistics/status) operation

A keyserver **MAY** support the "stats" operation.

The output of the "stats" operation is implementation-dependent, but may include diagnostic output, configuration state, or other metadata. The "search" variable is ignored when supplied with "stats".

4.1.5. The "hget" (hash get) operation

A keyserver **MAY** support the "hget" operation.

"hget" is used to search for a key by its digest rather than its key ID or fingerprint.

4.1.6. Other operations

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

4.2. 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.

4.2.1. 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 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 key ID **MUST** accept the 160-bit version 4 fingerprint and **MAY** accept 64-bit key IDs in the "search" variable. A keyserver **MUST NOT** return results for 32-bit "short key ID" searches, as these do not provide sufficient collision resistance.

4.2.2. 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 key ID or v4 fingerprint searches as it is not possible to calculate a key ID from a v3 fingerprint.

V3 keys are no longer considered secure, but **MAY** be distributed for historical reference.

4.2.3. Text Searches

How a keyserver handles a textual search is implementation defined. See also the definition of the "exact" variable (Section 6.5) for a method to give additional instructions to the server on how the search is to be executed.

4.3. Lookup Examples

Search for all keys containing the string "dshaw":

http://keys.example.com:11371/pks/lookup?search=dshaw&op=index

Get key 0xDEADBEEFDECAFBAD (64-bit key ID):

http://keys.example.com:11371/pks/lookup?op=get&search=0xDEADBEEFDECAFBA

5. Submitting Keys To A Keyserver

A keyserver **MAY** accept submissions via an HTTP POST request. Specifically, the abs_path (see [RFC1945], section 3.2) is set to "/pks/add", and the key data is provided via HTTP POST as specified in [RFC1945], section 8.3, and [RFC1866], 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 [RFC4880], sections 6.2 and 11.1. The ASCII armored keyring should also be urlencoded as specified in [RFC1866], section 8.2.1. Note that more than one key may be submitted in a single transaction.

There may also be modifier variables, as specified in Section 6 below.

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.

6. Modifier Variables

These variables are used to modify basic requests.

6.1. The "options" Variable

This variable takes one or more option values, separated by commas. These are used to modify the behavior of the keyserver on a per-request basis. Each value indicates a boolean flag, where the presence of the value indicates "true" and the absence "false".

6.1.1. 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 **SHOULD** be in machine-readable format. See Section 7 for the specific details of machine readable output.

6.1.2. 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 **MAY** trim any noncompliant user IDs before accepting the key. If this option was set, then such a key submission **SHOULD** fail with an appropriate error code such as 422 (Unprocessable content).

"nm" is meaningful for submissions only.

6.1.3. Other Options

Other site-specific or nonstandard options can be indicated by prefixing the option name with the string "x-". Non-standard options **MUST** represent boolean values with a default value of "false".

6.2. The "v" (Version) Variable

This variable identifies the version of machine readable output that the client supports. Currently, only "v=1" is defined.

"v" is meaningful for machine readable output only.

6.3. 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. An implementation **MAY** decide to ignore this variable or set the default behaviour to "on".

"fingerprint" is meaningful for lookups only.

6.4. The "hash" Variable

This variable takes one argument: "on" or "off". If present and on, it instructs the server to provide the [SKS] digest of each key in an "index" or "vindex" operation in the default human-readable form. 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. An implementation **MAY** decide to ignore this variable or set the default behaviour to "on".

"hash" is meaningful for lookups only.

6.5. 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. An implementation **MAY** decide to ignore this variable or set the default behaviour to "on".

"exact" is meaningful for lookups only.

6.6. Other Variables

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

7. Output Formats

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

A client implementation **SHOULD** request machine readable output and **SHOULD NOT** attempt to parse human-readable output.

7.1. Machine Readable Output

Clients requesting machine readable output:

* **SHOULD** supply "v=1" Section 6.2 and "option=mr" Section 6.1.1 in the request.

* **MUST** silently ignore any content preceding or following a returned armored key block.

* **MUST** silently ignore any keys with unknown versions or algorithms.

Keyservers returning machine readable output:

* **MUST** set the HTTP header "Access-Control-Allow-Origin: *", as specified in [CORS].

* **MUST** set "Content-Type: application/pgp-keys" when returning keys ("op=get"), as specified in [RFC3156], section 7.

*MUST use the format specified in Section 7.2 when returning indexes ("op=index").

*MAY return statistics in JSON format [RFC8259], the schema of which is otherwise implementation-dependent.

7.2. Machine Readable Indexes

The machine readable index format is a list of newline-separated records. The document is 7-bit clean, and as such is sent with no encoding and Content-Type: text/plain.

The machine readable response **MAY** be prefixed by an information record:

```
info:<version>:<count>
```

Field	Description
version	the version of this output format
count	the number of keys returned

Table 2: Information Record Fields

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

Note that "count" is the number of keys, and not the number of lines returned. That is, it **SHOULD** match the number of "pub:" lines returned.

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

```
pub:<keyid>:<algorithm>:<keylen>:<creationdate>:<expirationdate>:<flags>
```

Field	Description
keyid	the fingerprint or the key ID
algorithm	the algorithm ID
keylen	the key length in bits
creationdate	creation date of the key
expirationdate	expiration date of the key
flags	letter codes to indicate details of the key
version	the version of the key

Table 3: Public Key Record Fields

A keyserver **MAY** return a 16-digit key ID, but **SHOULD** return a fingerprint if available. Since it is not possible to calculate the

key ID from a V3 key fingerprint, for V3 keys this **SHOULD** be the 16-digit key ID only.

The algorithm ID is as specified in [RFC4880], section 9.1 i.e. 1==RSA, 17==DSA, etc.

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

```
uid:<uidstring>:<creationdate>:<expirationdate>:<flags>
```

Field	Description
uidstring	the user ID string
creationdate	creation date of the User ID
expirationdate	expiration date of the User ID
flags	letter codes to indicate details of the User ID

Table 4: User ID Record Fields

The user ID string **MUST** use HTTP % encoding for anything that isn't 7-bit safe as well as for the ":" character. Any other characters **MAY** be HTTP encoded, as desired.

The information for the "creationdate", "expirationdate", and "flags" fields is taken from the User ID self-signature, if any, and applies to the user ID in question, not to the key as a whole.

Primary key and User ID records may contain a "flags" field containing a sequence of alphabetical characters, one per flag. Flags **MAY** be given 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. Client implementations **MUST** ignore unknown flags.

Flag	Description
r	revoked
d	disabled
e	expired

Table 5: Record Flags

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. Client implementations

MUST ignore unknown trailing fields. All dates are given in the number of seconds since midnight 1/1/1970 UTC.

8. Locating a HKP Keyserver

Clients are usually manually configured with the address of a HKP keyserver. Client implementations 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 [RFC2782]. 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" when used over plaintext HTTP, or "hkps" when using HTTPS. 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. HKP clients **SHOULD** support SRV records.

8.1. Key discovery

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".

((andrewg : key discovery is the subject of ongoing debate, and may need to be left for another document. See issue #1.))

9. 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.

10. IANA Considerations

This document assigns the DNS SRV symbolic names "hkp" and "hkps", the URI schemes "hkp" and "hkps", and the HKP port 11371.

((andrewg : if we assign hkps, we may be required to specify a dedicated port, even though nobody uses it. See issue #14.))

11. References

11.1. Normative References

[CORS] "Cross Origin Resource Sharing", n.d., <https://fetch.spec.whatwg.org/#cors-protocol>.

[RFC1866] Berners-Lee, T. and D. Connolly, "Hypertext Markup Language - 2.0", RFC 1866, DOI 10.17487/RFC1866, November 1995, <https://www.rfc-editor.org/rfc/rfc1866>.

[RFC1945] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, DOI 10.17487/RFC1945, May 1996, <https://www.rfc-editor.org/rfc/rfc1945>.

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

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/rfc/rfc2782>.

[RFC3156] Elkins, M., Del Torto, D., Levien, R., and T. Roessler, "MIME Security with OpenPGP", RFC 3156, DOI 10.17487/RFC3156, August 2001, <https://www.rfc-editor.org/rfc/rfc3156>.

[RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007, <https://www.rfc-editor.org/rfc/rfc4880>.

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

[RFC9110]
Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

11.2. Informative References

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[SKS] "Synchronising Key Server Wiki", n.d., <https://github.com/sks-keyserver/sks-keyserver/wiki>.

Appendix A. Acknowledgments

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 authors 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.

Authors' Addresses

Daphne Shaw
Jabberwocky Tech

Email: dshaw@jabberwocky.com

Andrew Gallagher (editor)
PGPKeys.EU

Email: andrewg@andrewg.com