openpgp
Internet-Draft
Intended status: Standards Track
Expires: 18 September 2025

D. Shaw
Jabberwocky Tech
A. Gallagher, Ed.
PGPKeys.EU

17 March 2025

OpenPGP HTTP Keyserver Protocol
draft-gallagher-openpgp-hkp-07

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/.

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".

HKP(S) are distinguished from generic use of HTTP(S) by using the URI schemes "hkp" and "hkps" [RFC7595]. HKP is assigned port number 11371 and HKPS is assigned 11372 (although this is rarely used in practice). For reasons of maximum compatibility with firewalls and filtering HTTP proxies, HKP(S) are often served over the standard HTTP(S) port(s) (TCP ports 80 and 443).

By convention and history, HKP defaults to HTTP on TCP port 11371, and HKPS defaults to HTTPS on TCP port 443.

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

A keyserver SHOULD support both HKP and HKPS. A client SHOULD use HKPS, or a transport method with equivalent security properties, such as Tor hidden services [TOR].

See Section 10 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 under "/pks", but these are outside the scope of this document. These alternative paths have historically been used to provide human readable interfaces such as HTML forms, and functionality extensions such as [SKS].

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 RTBF
413 Content too large	The search returned too many responses
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.))

((andrewg : note also that 413 and 202 may be incorrect usage, see issues #25 and #26 respectively.))

4. Looking up Data from a Keyserver

Key lookups are done via an HTTP GET request. Specifically, the abs_path (Section 3.2 of [RFC1945]) is built up of the base path "/pks/lookup", followed by request-specific URL components. These components differ slightly between the Legacy and v1 request formats (see below).

Most HKP lookups contain both the "op" (operation) and "search" fields. The "op" field determines what operation the keyserver will execute, and the "search" field determines which keys are operated on.

There may also be modifier variables, as specified in Section 6 below. Variables are passed using HTTP query strings as specified in Section 8.2.2 of [RFC1866]. HTTP query strings MAY be given in any order. Keyservers MUST ignore any unknown query strings.

4.1. Legacy and v1 Request Formats

For backwards compatibility with the existing installed client base, a Legacy request format is defined. New implementations SHOULD use the v1 request format.

4.1.1. Legacy Request Format

In the Legacy request format, the "op" and "search" fields are supplied as HTTP query strings, in the form "<field-name>=<value>":

/pks/lookup?op=<op>&search=<search>[&...]

They are treated in the same way as modifier variables, including arbitrary ordering, however the "op" field MUST be supplied, and the "search" field MUST be supplied unless the "stats" operation is being requested (see Section 4.2.6). No URL path components under "/pks/lookup" are used.

4.1.2. v1 Request Format

In the v1 request format, the values of the "op" and "search" fields are supplied as URL path components. They are appended to "/pks/lookup/v1" as follows:

/pks/lookup/v1/<op>/<search>[?...]

The "op" and "search" fields MUST be supplied. Modifier variables are supplied as HTTP query strings, same as in the Legacy request format.

If the v1 request format is being used, machine readable output format (Section 7) MUST be returned. Note that the "stats" operation MUST NOT be used in v1 request format (see Section 4.2.6).

The v1 request format is designed so that a basic HKP service can be implemented using static files.

4.2. The "op" (operation) Field

The "op" field specifies the operation to be performed on the keyserver. The "op" field is generally used with the "search" field 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"). The server SHOULD NOT return an error code (such as 404 ("Not Found")) that could be interpreted by the client as an explicit statement of non-existence.

4.2.1. The "get" Operation

A keyserver SHOULD support the "get" operation.

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

The response to a successful "get" request is a HTTP document containing an ASCII-armored keyring as specified in Section 9.

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 [RFC9580].

A keyserver SHOULD limit the returned keyring to a reasonable length. Entries SHOULD be sorted in order of decreasing confidence (Section 8), and then by creation date (most recent first). If no keys match the request, the keyserver SHOULD return an appropriate HTTP error code such as 404 ("Not Found").

4.2.2. The "authget" (authoritative get) Operation

A keyserver SHOULD support the "authget" operation.

The "authget" operation is similar to the "get" operation, but is used for authoritative lookups by User ID only, in particular during key discovery (Section 11). The keyserver SHOULD only return results where the User ID being searched for has a nonzero confidence value (Section 8). If a keyserver is being used for key discovery, it MUST return only results for which it is has complete confidence. In addition, exact textual matching MUST be used even if the "exact" variable is set to "off" (Section 6.4).

4.2.3. The "since" Operation

A keyserver MAY support the "since" operation.

"since" requests a list of fingerprint prefixes that indicate which TPKs have been modified since midnight UTC on a specific date. The date is provided in the "search" field in ISO format with no time component, i.e. "2025-12-01".

The returned data is a list of CRLF-separated, hexadecimal-encoded fingerprint prefixes. The keyserver SHOULD calibrate the prefix length so that it is long enough to provide collision resistance, but short enough to maintain a useful anonymity cohort. A client MUST NOT make any assumptions about the length of the prefixes returned.

A client that wishes to update its local keyring from a keyserver MAY first make a "since" request with the date of the last successful refresh. It can then compare the returned list of prefixes to see if any of them are present in its local keyring, and make subsequent "vfpget" requests as appropriate (Section 4.2.7). In this way, it can avoid making unnecessary requests that will return no updates, but will still leak information to the keyserver.

Note that prefixes are always used, regardless of key version. This contrasts with key IDs, which are version-dependent.

A keyserver SHOULD NOT support lookup by prefix.

4.2.4. 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 in the "search" field. Historically, the "index" operation returned a human readable HTML document containing links for each found key, but this is not required.

A keyserver SHOULD limit the returned index to a reasonable length. Entries SHOULD be sorted in order of decreasing confidence, and then by creation date (most recent first). If no keys match the request, the keyserver SHOULD return an appropriate HTTP error code such as 404 ("Not Found").

4.2.5. 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 in the "search" field. Historically, a "vindex" response was the same as "index" with the addition of showing the signatures on each key, but this is not required. The "vindex" operation only differs from the "index" operation in Legacy human readable mode, and a server MAY treat the two operations as synonyms.

A server that supports "vindex" SHOULD treat it as a synonym for "index" in machine-readable mode. A client SHOULD NOT make machine-readable "vindex" requests.

4.2.6. The "stats" (statistics/status) Operation

A keyserver MAY support the "stats" operation in Legacy request format. The "stats" operation is deprecated, and MUST NOT be used in a v1 request. It is RECOMMENDED to use a URL outside "/pks/lookup" (such as "/pks/stats") instead.

The output of the "stats" operation is implementation-dependent, but may include diagnostic output, configuration state, or other metadata. The "search" field SHOULD NOT be supplied, and SHOULD be ignored if received.

4.2.7. The "vfpget" (versioned fingerprint get) Operation

A keyserver MAY support the "vfpget" operation.

"vfpget" requests a key from a keyserver by specifying its versioned fingerprint. The versioned fingerprint is provided in the "search" field in hexadecimal encoding, without a preceding "0x". The hexadecimal digits are not case sensitive.

A versioned fingerprint consists of one octet of key version number and N octets of fingerprint. This is the same octet sequence used in the Issuer Fingerprint and Intended Recipient Fingerprint subpackets (Section 5.2.3 of [RFC9580]).

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

4.2.8. The "kidget" (keyid get) Operation

A keyserver MAY support the "kidget" operation.

"kidget" requests a key from a keyserver by specifying its key ID (Section 5.5.4 of [RFC9580]). The key ID is provided in the "search" field as 16 hexadecimal digits, without a preceding "0x". The hexadecimal digits are not case sensitive.

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

4.2.9. The "hget" (hash get) Operation

A keyserver MAY support the "hget" operation.

"hget" requests a key from a keyserver by specifying its [SKS] digest. The digest is provided in the "search" field in hexadecimal encoding, without a preceding "0x". The hexadecimal digits are not case sensitive.

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

4.2.10. Other Operations

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

4.3. The "search" Field

The "search" field contains arbitrary text encoded as usual for a HTTP URL. This text may represent the key ID, or fingerprint, or some text from a user ID on the key being sought, depending on the operation.

4.3.1. Key ID and Fingerprint Searches

To search for a key by its key ID or fingerprint, a client SHOULD use a v1 request and either the "kidget" or "vfpget" operation (as appropriate).

If making a Legacy request, a client SHOULD use the "get" operation and prefix the "search" string with "0x" to indicate a hexadecimal number. Key ID strings are 16 hexadecimal digits (64 bits).

Fingerprint strings are either 32 (version 3), 40 (version 4), or 64 (version 6) hexadecimal digits. The hexadecimal digits are not case sensitive.

A keyserver:

* SHOULD accept fingerprints and MAY accept 64-bit key IDs in the Legacy request "search" field.

* MUST NOT return results for 32-bit "short key ID" searches, as these do not provide sufficient collision resistance.

* MUST NOT return v6 keys in the results for Legacy machine readable requests, but MAY do so for Legacy human readable requests (see also Section 7.2.1).

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

4.3.2. Text Searches

To search for a key by the text of a user ID, a client SHOULD use the "get", "authget", or "index" operation (as appropriate) and SHOULD NOT prefix the "search" string with "0x".

A keyserver MUST NOT return v6 keys in the results for Legacy machine readable requests, but MAY do so for Legacy human readable requests (see also Section 7.2.1). Otherwise, how a keyserver handles textual search is implementation defined. See also the definition of the "exact" variable (Section 6.4) for a method to give additional instructions to the server on how the search is to be executed.

4.4. Lookup Examples

Search for all keys containing the string "dshaw", using plaintext HTTP:

* Legacy request format:

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

* v1 request format:

  http://keys.example.com:11371/pks/lookup/v1/index/dshaw

Get key 0xDEADBEEFDECAFBAD (64-bit key ID), using HTTPS:

* Legacy request format:

https://keys.example.com/pks/lookup?op=get&search=0xDEADBEEFDECAFBAD

* v1 request format:

https://keys.example.com/pks/lookup/v1/kidget/DEADBEEFDECAFBAD

5. Submitting Keys To A Keyserver

A keyserver MAY accept submissions via an HTTP POST request. Specifically, the abs_path (Section 3.2 of [RFC1945]) is set to "/pks/add", and the key data is provided via HTTP POST as specified in Section 8.3 of [RFC1945], and Section 8.2.3 of [RFC1866].

The body of the POST message contains a "keytext" variable which contains an ASCII-armored keyring as specified in Section 9. The ASCII armored keyring should also be urlencoded as specified in Section 8.2.1 of [RFC1866]. 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. If the v1 request format (Section 4.1.2) is being used, machine readable output MUST be returned, so this option has no effect. 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 "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 in Legacy human readable output. In machine readable indexes, a value of "on" indicates that the "keyid" field SHOULD contain the fingerprint, except for v3 keys (see Section 7.2). An implementation SHOULD treat this variable as being "on" for all machine readable indexes. An implementation MAY decide to ignore this variable and/or set the default behaviour to "on" for Legacy human readable indexes.

"fingerprint" is meaningful for lookups only.

6.3. 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 Legacy human readable output format. This variable has no effect on any other operation, or on machine readable output. The exact format of the displayed digest, like the "index" and "vindex" operations themselves, is implementation defined. An implementation MAY decide to ignore this variable and/or set the default behaviour to "on".

"hash" is meaningful for lookups only.

6.4. The "exact" Variable

This variable takes one argument: "on" or "off". If set to "on", it instructs the server to search for an exact match for the contents of the "search" field. An implementation SHOULD set the default behaviour to "on" and MAY ignore this variable.

When "exact" is set to "on", a keyserver SHOULD only return results if the "search" field exactly matches one of the following:

* The full text string of a User ID.

* The portion between angle brackets ("<(.*)>") in an email-address style User ID.

In either case, the string matching SHOULD NOT be case sensitive.

"exact" is meaningful for lookups only.

6.5. Other Variables

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

7. Output Formats

HKP was originally intended for both human and programmatic use. In general, the Legacy human readable output is implementation specific. The "machine readable" option is used to tailor the output of Legacy requests for automated use. By contrast, the v1 request format MUST return machine readable output. 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 use the v1 request format, or else supply "options=mr" (Section 6.1.1) in a Legacy 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 (the "get", "vfpget", "kidget", and "hget" operations), as specified in Section 7 of [RFC3156].

* MUST use the format specified in Section 7.2 when returning indexes (the "index" and "vindex" operations).

* 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>:<keyversion>

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
keyversion	the version of the primary key

Table 3: Public Key Record Fields

Since it is not possible to calculate the key ID from a V3 key fingerprint, for V3 keys the "keyid" field SHOULD contain the 16-digit key ID only. Otherwise, a keyserver SHOULD return a fingerprint if available (see Section 6.2).

The algorithm ID is as specified in Section 9.1 of [RFC9580], 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
confidence	the confidence that the server has in this 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 ":" and "%" characters. 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. The "confidence" value is specified in Section 8.

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.

7.2.1. Legacy Machine Readable Indexes

For backwards compatibility with the installed client base, Legacy machine readable requests SHOULD return a restricted index format:

* The "keyversion" field and its preceding colon, and any subsequent fields, SHOULD be omitted.

* The "confidence" field and its preceding colon, and any subsequent fields, SHOULD be omitted.

* Version 6 (and later) keys SHOULD be omitted.

8. Confidence

Traditionally, keyservers did not perform any checks against uploaded content other than simple parseability. This left them open to abuse such as flooding and third-party signature spam. Most modern keyservers are now able to perform cryptographic validity tests, and automated content moderation is generally possible.

It is RECOMMENDED that a keyserver assigns a "confidence" value to each User ID in its database. The exact definition of "confidence" is implementation-dependent, but MAY include checks such as email deliverability or third-party certifications. A keyserver MAY publish its internal confidence value as a number between 0 and 255, where values of 120 or greater indicate "complete confidence". This is numerically compatible with the "trust amount" field specified in Section 5.2.3.21 of [RFC9580], but it is not required to calculate it in the same way or represent it internally as an integer value.

Keyserver confidence is intended as a heuristic for sorting and filtering responses to lookup requests, and is not a replacement for cryptographic verification. A client SHOULD NOT rely on the confidence value returned from a keyserver, and SHOULD perform its own checks.

9. Keyring Format

HKP uses an ASCII-armored keyring as its primary data representation for both input and output.

A keyring is a sequence of one or more OpenPGP Transferable Public Keys, concatenated directly, as specified in Sections 10.1 and 3.6 of [RFC9580]. An ASCII-armored keyring is a keyring that has been encoded as a single armored block, as specified in Section 6.2 of [RFC9580].

9.1. Detached Revocations

For OpenPGP keys prior to v6, revocation signatures have customarily been distributed as a detached "revocation certificate", as per Section 10.1.3 of [RFC9580]. An HKP server SHOULD allow submission of these detached revocations.

An HKP implementation MAY accept or serve an ASCII-armored "mixed keyring" where one or more revoked transferable public keys have been replaced by their detached revocation certificate(s). Such a "mixed keyring" MUST be sorted so that all detached revocation certificates appear first. This ensures that detached revocations cannot be mistaken for signatures over another key.

10. 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.

11. Key Discovery

Well-known URIs [RFC8615] are commonly used by clients to discover the location of internet services. We can use the "openpgpkey" well-known base path to locate an HKPS service for key discovery. This is done by serving an "hkps" discovery file that redirects a client to an authoritative HKPS service for the domain in question.

11.1. The "openpgpkey" Well-Known Base Path

[I-D.koch-openpgp-webkey-service] defines the well-known "openpgpkey" base path for a given domain as:

https://openpgpkey.<domain>/.well-known/openpgpkey/<domain>/

The "hkps" discovery file is placed under this directory, which is shared with other discovery methods such as WKD. The domain is repeated in a trailing path component in order to simplify serving multiple discovery files from a single host.

Some web hosting providers automatically return a zero length file successfully when a nonexistent file is requested, therefore:

* If the file "<base-path>/hkps" exists, and is of non-zero size, it SHOULD be used.

* If the file "<base-path>/hkps" does not exist, or is of zero size, then it MUST be ignored, and a client MAY fall back to other key discovery methods.

Requests for the discovery file MUST be made over HTTPS. The alternative "direct" well-known base path from [I-D.koch-openpgp-webkey-service] MUST NOT be used for HKPS discovery.

11.2. The "hkps" Discovery File

The discovery file contains a sequence of lines containing keywords and one or more values, separated by colons. Blank lines and comment lines beginning with "#" MUST be ignored.

The following keywords are defined:

version:<version>
server:<hostname>[:<port>[:<options>]]

If "port" is not given, it defaults to 443. "options" is a colon-separated list of options. No options are currently defined, and unknown options MUST be ignored. Unknown keywords MUST be ignored.

The discovery file is parsed by a client as follows:

* If a "version" line does not exist, a client MUST NOT continue with HKPS discovery, and MAY fall back to other discovery methods.

* If a "version" line exists but a "server" line does not, then the HKP service is located on the same hostname as the discovery file.

* If more than one "server" line exists, a client SHOULD try them in the given order, returning after the first successful response; a 404 ("Not Found") or 410 ("Gone") response in this context counts as a success.

Thus a minimal HKPS discovery file will contain just "version:1".

In addition:

* Only HKPS version 1 is supported in the discovery file, and clients MUST use the v1 request format and HTTPS for HKPS key discovery.

* If a client wishes to receive v6 or higher keys it SHOULD make an HKPS discovery request.

* If a server wishes to serve v6 or higher keys it SHOULD NOT do so over WKD, for compatibility with clients that do not support v6 keys.

11.3. Key Discovery Example

For example, a client trying to locate a key for isabella@silvie.example.com could consult:

```
https://openpgpkey.silvie.example.com/.well-known/openpgpkey/silvie.example.com/hkps
```

This discovery file might contain the following lines:

```
version:1
server:keyserver.example.com
```

On finding this file, a client SHOULD make a GET request for the following URL:

```
hkps://keyserver.example.com:443/pks/lookup/v1/authget/isabella@silvie.example.com
```

The domain owner MUST ensure that any keys returned from this keyserver are valid keys for the identity in the search field. This MAY be achieved at low cost by serving the keys from a static filesystem.

12. 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 arbitrary 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 arbitrary 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.

The sole exception to the above is if the openpgp key discovery method has been used, and all requests in the chain of indirection have been made over HTTPS/HKPS. In such a scenario, a client MAY assume that any keys returned are authoritative for the requested identity.

13. IANA Considerations

This document allocates the ports 11371 and 11372, the URI schemes "hkp" and "hkps", and the well-known URI suffix "openpgpkey".

13.1. Updated Registry Entries

IANA is requested to update the contact details for port 11371 in the "Service Name and Transport Protocol Port Number" registry to "Daphne Shaw".

13.2. New Registry Entries

IANA is requested to add the following entry to the "Service Name and Transport Protocol Port Number" registry as per [RFC6335]:

Service Name	Port	Transport Protocol	Description	Contact
hkps	11372	tcp and udp	OpenPGP HTTP Keyserver (Secure)	Daphne Shaw

Table 6: Service Name and Transport Protocol Port Number Registry

IANA is requested to add the following entries to the "URI Schemes" registry as per [RFC7595]:

URI Scheme	Description	Status	Reference
hkp	OpenPGP HTTP Keyserver	Permanent	This document
hkps	OpenPGP HTTP Keyserver (Secure)	Permanent	This document

Table 7: Uniform Resource Identifier (URI) Schemes Registry

IANA is requested to add the following entry to the "Well-Known URIs" registry as per [RFC8615]:

URI Suffix	Change Controller	Reference	Status	Related Information
openpgpkey	OpenPGP WG	This document	permanent	[I-D.koch-openpgp-webkey-service]

Table 8: Well-Known URIs Registry

14. References

14.1. 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>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/rfc/rfc7595>.

[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>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[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>.

[RFC9580] Wouters, P., Ed., Huigens, D., Winter, J., and Y. Niibe, "OpenPGP", RFC 9580, DOI 10.17487/RFC9580, July 2024, <https://www.rfc-editor.org/rfc/rfc9580>.

14.2. Informative References

[I-D.koch-openpgp-webkey-service]
Koch, W., "OpenPGP Web Key Directory", Work in Progress, Internet-Draft, draft-koch-openpgp-webkey-service-19, 5 December 2024, <https://datatracker.ietf.org/doc/html/draft-koch-openpgp-webkey-service-19>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/rfc/rfc6335>.

[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>.

[TOR] "Tor Specifications", n.d., <https://spec.torproject.org/>.

Appendix A. Acknowledgments

This document is a formalization and extension of 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 openpgpkey discovery file is based on previous work by Werner Koch, without whom the key discovery method would not exist in its current form. The "since" operation is based on an earlier proposal by Daniel Kahn Gillmor and Vincent Breitmoser.

The authors would 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. They would also like to thank Bart Butler, Daniel Huigens, Daniel Kahn Gillmor, and Justus Winter for help with the v1 request format and discovery file specifications.

Appendix B. Document History

Note to RFC Editor: this section should be removed before publication.

B.1. Changes Between draft-gallagher-openpgp-hkp-06 and draft-gallagher-openpgp-hkp-07

* Added "authget" and "since" operations.
* Defined confidence.
* Added more explicit guidance re sorting, filtering and error codes.
* Key version MR output field is now "keyversion" to distinguish from the output format version.

B.2. Changes Between draft-gallagher-openpgp-hkp-05 and draft-gallagher-openpgp-hkp-06

* Updated references.

B.3. Changes Between draft-gallagher-openpgp-hkp-04 and draft-gallagher-openpgp-hkp-05

* Allow detached revocations in keyrings.
* Redesigned v1 request format to use path components for required fields.
* Added openpgpkey discovery file.
* Added "vfpget" and "kidget" operations.
* Added meaningful "exact" semantics.
* HKPS is now SHOULD.
* Defined port 11372.
* IANA registry tables.
* Deprecated op=stats.

B.4. Changes Between draft-gallagher-openpgp-hkp-03 and draft-gallagher-openpgp-hkp-04

* Reworded Section 4 for clarity.
* Separate section for keyring format.
* Specify detached revocations.

* Updated references.

B.5. Changes Between draft-gallagher-openpgp-hkp-02 and draft-gallagher-openpgp-hkp-03

* Clients SHOULD supply the v=1 api-versioning variable.

* Machine-readable output includes key version field.

* Clients MUST silently ignore leading and trailing cruft, trailing unknown fields, and unknown flags.

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

* All other m-r index specs (CORS, Content-Type etc.) are now MUST.

* Included the hash variable from SKS.

B.6. Changes Between draft-gallagher-openpgp-hkp-01 and draft-gallagher-openpgp-hkp-02

* Tightened up BCP-14 language.

* Included op=hget from SKS.

* Options now strictly boolean with default false, variables less strict.

* More detail about HTTP status code usage.

B.7. Changes Between draft-shaw-openpgp-hkp-00 and draft-gallagher-openpgp-hkp-01

* Improved text structure.

* Added references to HTTPS/HKPS, and hkp:/hkps: URL schemes.

* Forbade short IDs and deprecated V3 keys.

* Included op=stats from SKS.

* Mentioned CORS.

* Made use of terminology more consistent.

* Replaced custom status codes with standard HTTP status codes.

Authors' Addresses

Daphne Shaw
Jabberwocky Tech
Email: dshaw@jabberwocky.com

Andrew Gallagher (editor)
PGPKeys.EU
Email: andrewg@andrewg.com