???
Internet-Draft
Intended status: Informational
Expires: 17 August 2025

F. Darling
T. Meunier
S. Newton
Cloudflare Inc.

13 February 2025

Recommendations for Key Directories over HTTP
draft-darling-key-directory-over-http-00

Abstract

This document defines recommendations for protocols that expose public keys over HTTP.

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://thibmeu.github.io/draft-darling-key-directory-over-http/draft-darling-key-directory-over-http.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-darling-key-directory-over-http/.

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

Source for this draft and an issue tracker can be found at https://github.com/thibmeu/draft-darling-key-directory-over-http.

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 17 August 2025.

Table of Contents

1. Introduction

Multiple Internet protocols rely on public key cryptography. They require keys to be distributed by Origins to Clients. This can be done via certificates, software releases, or HTTP. This document focuses on this last mechanism. It aims to set recommendations on how to design a key directory that is served over HTTP.

Distribution via HTTP allows for a more dynamic use of public keys, for rotation, or caching on intermediate mirrors or clients. This document specifies which cache directives origin should set, how clients and mirrors should consume cache directive set by origins, how origins should expose their key directory, and rotate them. This complements reccomendations provided by [HTTP-BEST-PRACTICES], narrowing them for key directories protocols.

The document does not cover a specific directory format, as these needs vary from one protocol to the next.

2. Motivation

Section 5 of [JOSE] and Section 7 of [COSE] both define ways to structure key sets, but no way to serve them. This creates issues when serving these keys over HTTP because caching is not taken into account, and there is no standard way to derive an ID from a key. Section 4 of [PRIVACYPASS], Section 3 of [OHTTP], and Section 4.7.1 of [DAP] are also defining their own public key directory, and are faced with similar issues. While Privacy Pass seems to have been the most thorough, even considering [CONSISTENCY] for instance, these seem to be duplicated efforts that would benefit from being consolidated into one specification.

3. Presentation Language

This document uses the TLS presentation language [RFC8446] to describe the structure of protocol messages. In addition to the base syntax, it uses two additional features: the ability for fields to be optional and the ability for vectors to have variable-size length headers.

3.1. Variable-Size Vector Length Headers

In the TLS presentation language, vectors are encoded as a sequence of encoded elements prefixed with a length. The length field has a fixed size set by specifying the minimum and maximum lengths of the encoded sequence of elements.

In this document, there are several vectors whose sizes vary over significant ranges. So instead of using a fixed-size length field, it uses a variable-size length using a variable-length integer encoding based on the one described in Section 16 of [RFC9000]. They differ only in that the one here requires a minimum-size encoding. Instead of presenting min and max values, the vector description simply includes a V. For example:

```
struct {
    uint32 fixed<0..255>;
    opaque variable<V>;
} StructWithVectors;
```

4. 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.

The following terms are used throughout this document:

Client: An entity using public key material.

Origin: An entity exposing public key material via HTTP.

Mirror: An intermediary entity between client and origin. May cache data, and act as a privacy proxy.

Key metadata: Public data associated to a public key.

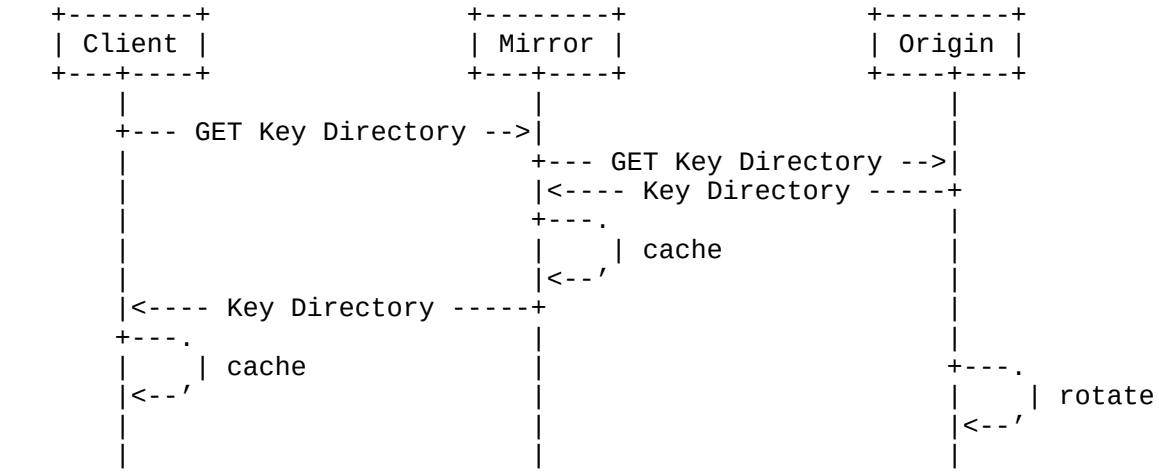
Key Directory: Set of public keys.

Directory Metadata: Public data associated to a key directory. This is protocol specific.

5. Architecture

An Origin is exposing a key directory for Clients to fetch. Clients may fetch the directory from a Mirror, either to protect its privacy, or because the Origin wants to leverage a content delivery network. The endpoint and request pattern should be the same as if the fetch was to an Origin.

This document focuses on the below interaction, which is triggered when the Client does not have valid key for the Origin. This can be because the Client is new, its cache is expired, or the Origin refuses requests with the current key set.



5.1. Key ID

Each key in the directory is associated with a unique Key ID.

Key ID has to be derived from key material that is shared publicly. Protocols SHOULD provide the following blob of data:

```
struct {
  opaque ProtocolBlob<V>;
} PublicKeyMaterial;
```

PublicKeyMaterial may be composed of both cryptographic material and metadata.

Key ID is defined has follow:

key_id = encode(H(PublicKeyMaterial))

where

* PublicKeyMaterial is a length-prefix-encoded blob of data

* H is a hash function

* encode is some encoding function

TODO Open questions about H

* Should the draft provide specific H?

* Should the draft define an IANA registry and require protocols to register their H?

5.2. Key Selection

The following is a deterministic algorithm for determining which Key a Client uses to fulfill their cryptographic needs. By using a deterministic algorithm, Origins can more easily predict the effects of a Key rotation and implement grace periods, soak times, etc. Protocols may place additional restrictions, or push these decision details to deployments.

5.2.1. Algorithm

1. *Filter invalid keys*: Exclude keys that:
    * Have a not-after field in the past.
    * Have a Section 5.4.1 field in the future.
    * Do not meet required cryptographic properties.
2. *Set missing activation times*: If a key does not have a Section 5.4.1 field, set it to either:
    * The Last-Modified header from the request as defined in Section 8.8.2 of [HTTP], if available.
    * The Date header from the request as defined in Section 6.6.1 of [HTTP], if available.
    * The client's local time.
3. *Sort by activation time*: If a Section 5.4.1 field exists, sort the remaining keys in *descending* order based on Section 5.4.1.

4. *Select the first key*: Choose the first key from the Key Directory, as ordered in the Key Directory format.

Clients should implement the Key Selection Algorithm. Origins should present the newest Keys first.

For protocols which define a Section 5.4.1 field, the above algorithm minimizes the chance that the Client uses a key that has expired between fetching the directory from the origin and its usage as part of the protocol.

For protocols without a Section 5.4.1 field, using the first key allows Origin to present their key directory so that the newest is always first, and the soon-to-be-removed key is last. This minimizes the chance of a client using an expired key.

Expired key may be presented for completion, only if the protocol defines a not-after field.

5.3. Rotation

Key directories are not permanent: they change over time. Origins SHOULD rotate keys on a schedule. Clients are going to fetch keys upon an immediate rotation for security reasons. This section goes over how an Origin rotates its keys, and how that interacts with scheduled and immediate rotations.

5.3.1. Algorithm

We approach a public key generation by the following function

Generate(params, RAND) -> (publickey, privatekey, metadata)

At any point in time, all keys in the directory have a unique key id as defined in Section 5.1. When adding a key in the directory, that key has a unique key id.

Generation looks as follows

```
do
  (publickey, privatekey, metadata) <- Generate(params, RAND)
  public_key_material <- (publickey | metadata)
  key_id <- encode(H(public_key_material))
while (key_id is not unique)
```

5.3.2. Scheduled rotation

Scheduled rotation happens at a time known to Origins, Clients, and Mirrors. This may be a regular interval (monthly, weekly, daily), or an ad-hoc schedule agreed between all parties.

Scheduled rotations are communicated in one of the two mode below

Passive: Origins rely on cache headers to inform Clients about key expiry. They stop advertising the key at time t, and delete it at time t_expiry=t+maxage. Origins may have to take intermediate mirrors into considerations, if they are aware these mirrors don't respect their cache headers.

Active: Origins keep serving the key and add a not-after field. This field should be at least t+maxage.

With both modes, an Origin has to signal a key is not supported by sending a response with status code 400. For instance, the error can include a key ID as defined in Section 5.1.

5.3.3. Immediate

Origins might have to rotate keys immediately. Existing keys may have to be invalidated and/or new keys be provisioned. Immediate key rotation can happen in the event of a key compromise, loss, or other imperious reason.

Immediate key rotation will cause some client requests to the server to fail until the client and mirrors retrieve a new version of the directory. The key directory endpoint is going to be placed under a higher load.

1. Origins should consider introducing a random backoff to spread the load of key distribution over time. See Section 5.4

2. Clients on a scheduled rotation should be configured to distrust rotation outside a fixed window. Such policies should be defined by protocols.

5.4. Cache behaviour

Caching the Key Directory lowers latency and reduces resource usage on Mirrors and the Origin. An optimal caching strategy should minimize resource usage for both the Client and Origin while preventing the client from using an invalid key.

These two requirements, minimizing resource usage and never using an invalid key, are at odds with each other. In the event of an Section 5.3.3 key rotation, a Client might use an invalid key. However, if a Client fetches an Origin key directory for every request, it would waste time and network resources.

This section provides interaction with some cache directive. Deployments should also consider the recommendations laid out in Section 4.9 of [HTTP-BEST-PRACTICES].

5.4.1. not-before fields

Protocols should define a not-before field. This field can be attached by Origins to each Key in the Directory. The not-before field allows Origins to signal to Clients when a Key is ready to use and reduces the chance a Client uses a key which is not yet available. not-before fields SHOULD be a Unix epoch timestamp in seconds.

5.4.2. Cache directives

Origins responds with cache directives [HTTP-CACHE]. These control when the Key Directory should be refreshed. For instance, Origins provides a Cache-Control: max-age header, or Expires header which is slightly less than the grace period given for a key about to rotate. Clients should respect the max-age cache directive or other directives. If an Origin provides a max-age header and a Mirror is used, an Origin should provide a s-maxage header that is equivalent to max-age.

To prevent Clients refreshing their Key Directories at the same time (synchronization), Mirrors should provide to its clients a max-age cache directive with duration in the range [0, Origin s-maxage].

Origins should consider using Date (Section 6.6.1 of [HTTP]) and Last-modified (Section 8.8.2 of [HTTP]) headers to ease Section 5.3.

5.4.3. Client cache refresh

The primary method a Client should use to determine when it refreshes its view of the Key Directory is through the delta seconds described in the max-age cache directive. The higher the delta, the less frequent a Client will update its cache. The lower the delta, the quicker clients will respond to unplanned key rotations.

min-fresh could also be sent by Origins as defined in [HTTP-CACHE].

5.5. Well known URL

It is recommended protocols register a [WELL-KNOWN] URL and associated content-type.

A key directory server should consider supporting both GET and HEAD request on that endpoint.

```
GET /.well-known/<your-protocol>

HTTP/1.1 200 OK
Cache-Control: max-age=<Client Cache TTL>, s-maxage=<Shared Cache TTL>
Content-Type: <your-protocol>
Last-Modified: <datestamp>
```

HEAD requests can be used by Clients to cheaply determine if the directory has changed. The Origin server SHOULD issue a Last-Modified header with the date stamp of when the key directory resource was last modified.

If issuing a Last-Modified header, the Origin server SHOULD support the correct response to a If-Modified-Since HTTP GET or HEAD request, returning the appropriate HTTP status codes [HTTP-CACHE].

It is recommended that Mirrors support Last-Modified and If-Modified-Since [HTTP-CACHE].

5.6. Future considerations

These considerations should be addressed in future drafts. Defining them now appears to be premature as the core of the draft does not have consensus.

5.6.1. Consistency

Consistency allows client to prevent themselves from split view attack. A proposal that has been made for Privacy Pass is to use multiple mirrors [CONSISTENCY]. With a sufficiently high quorum, clients get more confident that they are not singled out. It presents scalability issues as you need multiple mirrors, and have one more requests from client per mirror in the quorum.

5.6.2. Key Transparency

Key Directory over HTTP SHOULD integrate with transparency, once the protocol has been defined in [KEYTRANS]. There are specific consideration as to what goes in the log: the full directory, keys individually, privacy considerations.

6. Privacy Considerations

Clients fetching keys mean they reveal their IP, time, and other informations. When the key directory is provided by an Origin distinct from the Origin providing service, Clients SHOULD consider proxying their traffic through a Mirror server provided by the service they use. This may happen if the Origin service delegates key management to a third party for instance. Mirrors SHOULD NOT collide with the key server in an attempt to break Client privacy.

7. Security Considerations

TODO Security

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. Normative References

[HTTP] 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>.

[HTTP-CACHE] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/rfc/rfc9111>.

[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>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

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

9.2. Informative References

[CONSISTENCY]
Beurdouche, B., Finkel, M., Valdez, S., Wood, C. A., and T. Pauly, "Checking Resource Consistency with HTTP Mirrors", Work in Progress, Internet-Draft, draft-ietf-privacypass-consistency-mirror-00, 30 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-consistency-mirror-00>.

[COSE] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/rfc/rfc8152>.

[COSE-HPKE]
Tschofenig, H., Steele, O., Daisuke, A., and L. Lundblade, "Use of Hybrid Public-Key Encryption (HPKE) with CBOR Object Signing and Encryption (COSE)", Work in Progress, Internet-Draft, draft-ietf-cose-hpke-10, 18 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-hpke-10>.

[DAP] Geoghegan, T., Patton, C., Pitman, B., Rescorla, E., and C. A. Wood, "Distributed Aggregation Protocol for Privacy Preserving Measurement", Work in Progress, Internet-Draft, draft-ietf-ppm-dap-14, 3 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ppm-dap-14>.

[HTTP-BEST-PRACTICES]
Nottingham, M., "Building Protocols with HTTP", BCP 56, RFC 9205, DOI 10.17487/RFC9205, June 2022, <https://www.rfc-editor.org/rfc/rfc9205>.

[JOSE] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[JOSE-HPKE]
Reddy.K, T., Tschofenig, H., Banerjee, A., Steele, O., and M. B. Jones, "Use of Hybrid Public Key Encryption (HPKE) with JSON Object Signing and Encryption (JOSE)", Work in Progress, Internet-Draft, draft-rha-jose-hpke-encrypt-07, 31 March 2024, <https://datatracker.ietf.org/doc/html/draft-rha-jose-hpke-encrypt-07>.

[KEYTRANS] McMillion, B. and F. Linker, "Key Transparency Protocol", Work in Progress, Internet-Draft, draft-ietf-keytrans-protocol-00, 10 December 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-keytrans-protocol-00>.

[OHTTP] Thomson, M. and C. A. Wood, "Oblivious HTTP", RFC 9458, DOI 10.17487/RFC9458, January 2024, <https://www.rfc-editor.org/rfc/rfc9458>.

[PRIVACYPASS]
Celi, S., Davidson, A., Valdez, S., and C. A. Wood, "Privacy Pass Issuance Protocols", RFC 9578, DOI 10.17487/RFC9578, June 2024, <https://www.rfc-editor.org/rfc/rfc9578>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

Appendix A. Test vectors

Implementation may test the following scenarios 1. One key is provided on a normal basis, the key rotates: 1. does the directory has two entries or one? 2. what are the cache headers? 3. what are the keys as seen by a client? 4. does the new key has a unique key id?

1. Key has to be rotated immediately

2. How long does it take for client to notice

3. What is the backoff to avoid a thundering herd issue

Appendix B. Non normative proposals

B.1. For JOSE

JOSE has a key directory in the form of JWK Set as defined in Section 5 of [JOSE]. However, it does not addhere to the best practices laid down here, despite being Web keys. This would be valuable as the group aims to integrates with all sort of cryptographic keys, as can be seen with the new HPKE proposal [JOSE-HPKE].

Following these recommendations, JOSE may define:

* A well-known URI

* A deterministic Key ID

* Recommendations for caching and rotation, or leave it to implementations.

B.2. For COSE

Same as the above, for COSE Key Set.

Appendix C. Use cases

See existing key directory on https://key-directory-over-http.research.cloudflare.com/

C.1. DAP

Defined in Section 4.5.1 of [DAP].

```
HpkeConfig HpkeConfigList<0..2^16-1>;

struct {
  HpkeConfigId id;
  HpkeKemId kem_id;
  HpkeKdfId kdf_id;
  HpkeAeadId aead_id;
  HpkePublicKey public_key;
} HpkeConfig;

opaque HpkePublicKey<0..2^16-1>;
uint16 HpkeAeadId; /* Defined in [HPKE] */
uint16 HpkeKemId;  /* Defined in [HPKE] */
uint16 HpkeKdfId;  /* Defined in [HPKE] */
```

Partially informed comments:

* HpkeConfigId could be removed

* Need not-before to handle early capture

C.2. OHTTP

Defined in Section 3 of [OHTTP].

```
HPKE Symmetric Algorithms {
  HPKE KDF ID (16),
  HPKE AEAD ID (16),
}

Key Config {
  Key Identifier (8),
  HPKE KEM ID (16),
  HPKE Public Key (Npk * 8),
  HPKE Symmetric Algorithms Length (16) = 4..65532,
  HPKE Symmetric Algorithms (32) ...,
}
```

Partially informed comments:

* Key Identifier could be removed/be deterministic

* No mention of not-before

* No mention of HTTP Caching for rotation

C.3. Privacy Pass

Defined in Section 4 of [PRIVACYPASS]

```
 {
    "issuer-request-uri": "https://issuer.example.net/request",
    "token-keys": [
      {
        "token-type": 2,
        "token-key": "MI...AB",
        "not-before": 1686913811,
      },
      {
        "token-type": 2,
        "token-key": "MI...AQ",
      }
    ]
 }
```

Partially informed comments: * Not as flexible as HPKE * Has some protocol metadata (token-type, issuer-request-uri, rate-limit)

Acknowledgments

The authors would like to thank Ethan Heilman, Michael Rosenberg, and Chris Wood for their knowledge of existing practices on key directories, and Mark Nottingham for helpful discussion on HTTP best practices.

Authors' Addresses

Fisher Darling
Cloudflare Inc.
Email: fisher@darling.dev

Thibault Meunier
Cloudflare Inc.
Email: ot-ietf@thibault.uk

Simon Newton
Cloudflare Inc.
Email: rfc@simonnewton.com