Workgroup: openpgp
Internet-Draft: draft-nwjw-openpgp-cert-d-00
Published: 31 May 2022
Intended Status: Informational
Expires: 2 December 2022
Authors: N. Widdecke J. Winter
Sequoia PGP Sequoia PGP

Shared OpenPGP Certificate Directory

Abstract

This document defines a generic OpenPGP certificate store that can be shared between implementations. It also defines a way to root trust, and a way to associate petnames with certificates. Sharing certificates and trust decisions increases security by enabling more applications to take advantage of OpenPGP. It also improves privacy by reducing the required certificate discoveries that go out to the network.

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://sequoia-pgp.gitlab.io/pgp-cert-d. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-nwjw-openpgp-cert-d/.

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/.

Source for this draft and an issue tracker can be found at https://gitlab.com/sequoia-pgp/pgp-cert-d.

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 December 2022.

Copyright Notice

Table of Contents

1. Introduction

Using OpenPGP for encryption requires a certificate for each communication partner. Likewise, verification of an OpenPGP signature requires the signer's certificate.

An OpenPGP certificate must be discovered before it can be used. There are a number of ways to do that, for example via [keys.openpgp.org] or [I-D.draft-koch-openpgp-webkey-service-12].

Furthermore, an OpenPGP certificate evolves over time. The certificate itself or one of its components may be revoked; a User ID may be added; certificate subkeys may be rotated, and meta-data stored on signatures updated. Crucially, the security of OpenPGP depends on distributing each update to every involved party. A certificate update may be passively collected (e.g. by consuming an [Autocrypt] header), or actively sought out using the key discovery options mentioned above.

However, actively reaching out to a network source leaks information about the expected communication partner or partners, so requests should be kept to a minimum. Now, if a user has more than one application supporting OpenPGP, then every application has to discover certificates and updates, increasing the meta-data leakage. The obvious solution here is to provide a way to share the certificates instead. This is the purpose of this specification.

Looking at X.509, we can see that on most systems, there is a shared store of root certificates. Now, this root certificate store solves a different problem: X.509 certificates do not need to be

discovered. Instead, the shared store ensures that every application uses the same set of trust roots, which is also desirable for OpenPGP. The important aspect we want to point out is that the store is shared across different applications and TLS implementations. We will come back to the differences later in this text.

1.1. Requirements Language

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.

1.2. Terminology

This document uses the term "key" to refer exclusively to OpenPGP Transferable Secret Keys (see section 11.2 of [RFC4880]).

It uses the term "certificate", or "cert" for short, to refer to OpenPGP Transferable Public Key (see section 11.1 of [RFC4880]).

1.3. Related work

1.3.1. OpenPGP keyrings

The classic way of sharing data between OpenPGP implementations is via a keyring (see section 3.6 of [RFC4880]). The only defined format is simply a sequence of certs, stored in binary (not ASCII-armored) format. The advantage is that only OpenPGP data structures are used, and hence support for keyrings is widespread in OpenPGP implementations.

But, because an OpenPGP keyring does not have an index, this data structure scales badly: Both lookups and updates take O(N) time, where N is the number of certs. Worse, if the keyring contains a flooded certificate, it will negatively affect the performance of every operation, not just operations on the flooded cert itself.

Because of these limitations, an OpenPGP keyring as defined in [RFC4880] should really only be used as interchange format (i.e. for import and export), not for continuous sharing of certs between applications and implementations.

1.3.2. X.509 certificate stores

Looking at X.509, we can see that on most systems, there is a shared store of root certificates (see e.g. [FedoraSharedX.509CertStore], [WindowsSharedX.509CertStore], [macOSSharedX.509CertStore]). Now,

this root certificate store solves a slightly different problem: It ensures that every application uses the same set of trust roots.

During the Transport Layer Security ([TLS]) handshake, the party that wants to authenticate itself (usually the server) presents the certificate, along with all intermediate certificates in the authentication chain up to a root certificate. In this setting, we don't need to discover any certificates. Instead, the store is used to check if the presented root certificate is in the set of trusted root certificates. Additionally, the store may contain certificate revocations.

In both OpenPGP and X.509, trust must be rooted. While the predominant trust model in X.509 uses a fixed set of vendor-specified trusted third parties, in OpenPGP the user is expected to provide this set. See Section 2.4 for how this is modeled in this spec.

The main takeaway here is that to ensure a consistent behavior and user experience, the certificate store with all its information should be shared across all applications that use OpenPGP to authenticate communication partners.

1.3.3. Maildir

Maildir is an on-disk data structure that is designed to allow concurrent access by programs storing mails into and retrieving them (see [Maildir]). It allows lock-free operations by relying on the atomicity of rename(2). It is supported by a wide range of mail servers, delivery agents, and mail user agents.

Maildir is mainly concerned with storing blobs and orchestrating concurrent access to the store. It does not provide any indices. For example, if you need efficient full text search, you will need to construct an index on top of the maildir, and keep it up to date (see [Notmuch]).

Maildir's design and success is a major inspiration for this spec.

2. Requirements

This specification is motivated by the following requirements:

* The performance should not be affected by the number of certificates in the store, or by the size of individual certificates.

* We expect a read-heavy workload. As such, readers should not have to synchronize with each other or with writers.

- *Updates must be robust, i.e., they must not lose information in the event of concurrent updates.

- *No extra data structures besides the file system and OpenPGP should be used to facilitate adoption by OpenPGP implementations.

Furthermore, the following requirements are required for secure and ergonomic use of OpenPGP. Since any application using OpenPGP needs to behave consistently so as not to jeopardize security and ergonomics, this information needs to be shared well. Hence the ideal place is the certificate store:

- *Address book-like mapping from petnames to certificates.

- *Configuring a set of trusted introducers.

We also like to have some non-functional properties:

- *The data structure should be efficient to backup.

- *The data structure should be efficient to synchronize between machines.

2.1. Addressing of certs

Conceptually, the cert store is a name-value store. We use cert fingerprints as names, as well as a set of special names. This is accomplished by mapping names to paths, then relying on the filesystem for efficient lookups.

2.2. Trust root

One certificate in the store is used to root trust. It is used for mapping petnames to certs (see Section 2.3), and to designate trusted introducers (see Section 2.4).

2.3. Petname mapping

Petnames span a namespace that is secure and human-meaningful, but not distributed. A common example of a petname scheme are address books in mobile phones that securely map human-meaningful names to numbers (which are secure and distributed, but not human-meaningful). See [Zookos-Triangle] for a more in-depth discussion.

Using petnames, we can securely map human-meaningful names, like "Mom" or "juliett@example.org", to OpenPGP certificates. In contrast to many other trust models, this is a concept that most users are already familiar with. Therefore, it should be easy to train users, increasing the chance that they will use it in a secure manner.

The petname mapping can also be used to integrate into existing address book-like functionality provided by the platform.

2.4. Trusted introducers

To improve the ergonomics of public-key systems, users often delegate questions about the identity of a communications partner to some set of trusted third parties.

In X.509, these decisions are delegated to a fixed set of vendor-specified trusted third parties known as root certification authorities (see [X509-PKI]). These trusted third parties then usually certify intermediate certification authorities, which in turn certify the binding between a peer's key and its identity. The trust relation forms a polyforest (i.e., a directed graph with multiple roots).

Using this trust relation as a client during the [TLS] handshake is straightforward: The server presents its certificate along with the chain of all intermediate certificates up to the root. The client simply checks if all links in the chain are valid, and whether the terminal certificate is in its set of root certification authorities. If so, the server's certificate is authenticated.

In contrast, the trust relation in OpenPGP forms a directed graph. Any certificate can certify that a cert belongs to an identity. Furthermore, the user is expected to provide not only the set of trust roots (the equivalent of X.509's root certification authorities), but also to identify acceptable intermediate authorities, which are known as "trusted introducers" in OpenPGP parlance.

Traditionally, OpenPGP implementations have used idiosyncratic mechanisms to configure both the trust roots and the trusted introducers. That has the downside of being a proprietary mechanism that cannot easily be shared between implementations. In contrast, this specification uses a single distinguished certificate as a trust root that delegates authority to the trusted introducers.

3. Implementation

This section describes in detail how to interact with a certificate store. Note that we also provide a library that abstracts this away behind a simple-to-use API.

3.1. Default store's location

If not explicitly requested otherwise, an application **SHOULD** use the default store. The location is platform specific, see Section 3.8 for details.

The default store may be overridden by the user by setting the environment variable PGP_CERT_D.

The application may explicitly choose to use a different location entirely. Note, however, that this should be done only with good reasons, because it jeopardizes security, privacy, and ergonomics.

The location of the store **MUST** be a directory. If it does not exist, it **MAY** be created on demand.

3.2. Mapping names to paths

Names are either fingerprints or special names.

3.2.1. Fingerprints

The store is indexed by fingerprint. This is achieved by using the file system as a dictionary, storing each certificate using a path derived from the cert's fingerprint.

To compute the path to the certificate file:

- compute the cert's fingerprint,
- format it using lowercase hex digits,
- take a two-digit prefix as sub-directory name,
- use the remaining digits as the filename.

For example, the certificate with the fingerprint eb85bb5fa33a75e15e944e63f231550c4f47e38e will be stored at ${BASEPATH}/eb/85bb5fa33a75e15e944e63f231550c4f47e38e.

3.2.2. Special names

There is a set of special names that can be used to address certificates in the store. The names map to fixed locations in the store.

Special name	Location
trust-root	trust-root

Table 1

3.3. Locking the store for writes

Before a cert can be inserted or updated, you **MUST** acquire an exclusive lock on the store. Note that this lock only synchronizes writers: Concurrent readers can continue to use the store, and will always see consistent certs.

The lock to the store is represented by a file located at ${BASEPATH}/writelock which does not contain any data. If that file does not exist, the store **SHOULD** be assumed unlocked and the file **MUST** be created before any locking operation. The locking is achieved with file descriptors using platform specific means, see Section 3.8 for details.

3.4. How to insert or update certs

The following procedure **MUST** be followed to ensure that concurrent readers are not disturbed:

* First, acquire an exclusive lock. See Section 3.3.

* Then, look up the cert you want to insert or update in the store.

* If the store contains a copy of the cert, merge it with your copy.

* Write the cert to a temporary file. This file **MUST** be on the same filesystem if the platform requires this for atomic replacement in the next step (e.g. on POSIX, rename(2) fails if the rename crosses filesystem boundaries).

* Atomically replace the existing cert with the temporary file (i.e. using rename(2) on POSIX).

* Release the exclusive lock.

If a certificate is stored using a fingerprint as name, the name **MUST** match the certificate's fingerprint.

3.5. Rooting trust

3.5.1. Trust root

The trust root is an OpenPGP certificate that is stored under the special name trust-root.

The certificate:

* **MUST** be certification capable.

* **SHOULD** have a User ID to increase compatibility.

* **SHOULD NOT** have any subkeys.

* **SHOULD** use direct key signatures or binding signatures that are marked as non-exportable.

*MAY have a secret key, password protected or not.

If the certificate has a secret key, then any conforming OpenPGP implementation can use it to add a petname or a trusted introducer. Otherwise, only an implementation with access to the secret key material can do so.

3.5.2. Petname mapping

To add a petname to a certificate, create a User ID with the desired petname, and bind it to the target certificate using the trust root. The binding signature **SHOULD** be marked as non-exportable.

To remove a petname from a certificate, revoke the User ID using the trust root. The revocation signature **SHOULD** be marked as non-exportable.

To look up certificates by petname, iterate over the store returning all certificates that contain the petname as User ID bound by the trust root.

This lookup **SHOULD** be facilitated using an index data structure. Currently, we do not define such an index structure, but we define an extension mechanism so that the index can be stored in the store (see Section 3.6).

3.5.3. Trusted introducers

To mark a certificate as trusted introducer, create a direct key signature for the trusted introducer using the trust root, with a subpacket marking it as trust signature. The trust signature **MAY** be scoped. The signature **SHOULD** be marked as non-exportable.

To rescind a trust delegation, create a new direct key signature for the trusted introducer using the trust root, without a subpacket marking it as trust signature. The signature **SHOULD** be marked as non-exportable.

The trust root can be used in conjunction with the default OpenPGP trust model to authenticate nicknames attached to certificates. To look up certificates by nickname, explore the trust relation of certs in the store starting with the trust root. Return all certificates that contain the desired nickname as User ID which are corroborated by a path from the root to the certificate.

This lookup should be facilitated using an index data structure. Currently, we do not define such an index structure, but we define an extension mechanism so that the index can be stored in the store (see Section 3.6).

3.6. Proprietary and experimental extensions

Files or directories in the toplevel directory starting with an underscore (_) may be freely used for proprietary and experimental extensions. Please use a unique and descriptive prefix to minimize the chance of collisions, e.g. _foopgp_subkey_map.sqlite.

Unknown or unsupported extensions **MUST** be ignored.

3.7. Reserved filenames

Any files or directories in the toplevel directory other than

*fingerprints mapped to paths (see Section 3.2)

*known special names mapped to paths (see Section 3.2.2)

*files starting with an underscore (_) (see Section 3.6)

are reserved for future extensions and **MUST** be ignored.

3.8. Platform-specific conventions

Platform	Default store location	Locking mechanism
POSIX	$XDG_DATA_HOME/pgp.cert.d	flock(2) with LOCK_EX
macOS	$HOME/Library/Application Support/ pgp.cert.d	flock(2) with LOCK_EX
Windows	{FOLDERID_RoamingAppData}/pgp.cert.d	LockFile (fileapi.h)

Table 2

4. Examples

Importing the certificates described [I-D.draft-bre-openpgp-samples-00] yields the following certificate store:

```
$ export PGP_CERT_D=$(mktemp -d)
$ pgp-cert-d import < alice.pgp
$ (cd $PGP_CERT_D ; find -type f)
./eb/85bb5fa33a75e15e944e63f231550c4f47e38e
$ pgp-cert-d import < bob.pgp
$ (cd $PGP_CERT_D ; find -type f)
./eb/85bb5fa33a75e15e944e63f231550c4f47e38e
./d1/a66e1a23b182c9980f788cfbfcc82a015e7330
```

5. Reference implementation

We provide a reference implementation in the form of a library implemented in Rust (see [reference-implementation-api]). This library also has a C API, so it is easy to use from other languages.

The library deals with the low-level mechanics of accessing the store, and computing the fingerprints of inserted certs. It does not concern itself with emergent features like petname and authenticated nickname lookups.

5.1. Opening the store

There are two ways to open a store. The first one uses the default location, the second takes a path to the store's location.

```
function new() -> Store;
function open(Path) -> Store;
```

5.2. Certificate lookup

Looking up a certificate returns the certs data and a tag if the certificate exists in the store, or a special value indicating that the cert was not found.

The tag can be used in subsequent lookups to quickly check if the cert has actually changed. This can be used to efficiently update index data structures.

Usually, this function returns a CERT (Section 5.5.3), but if NAME (Section 5.5.1) is a special name, it may return a KEY (Section 5.5.4).

```
function Store::get(NAME) -> Maybe(TAG, CERT-or-KEY);
function Store::get_if_changed(TAG, NAME) -> Maybe(TAG, CERT-or-KEY);
```

5.3. Certificate update

Inserting or updating a cert requires the CERT (Section 5.5.3) and a callback function.

The callback is invoked with the existing cert data (if any), and **SHOULD** merge the two copies of the certificate together. The function **MAY** decide to omit (parts of) the existing data, but this should be done with great care as not to lose any vital information.

The insertion method returns the merged certificate data and the tag for the new state.

Locking is handled by the library.

```
function Store::insert(CERT, Merge) -> (TAG, CERT)
    where Merge is
        function(CERT, Maybe(CERT)) -> CERT;
```

5.4. Store enumeration

The user can iterate over all certificates in the store. The iterator returns tuples of fingerprints and tags, which can be used to efficiently update index data structures.

Note: The iterator does not return any special names like the trust root (see Section 3.2.2).

```
function Store::iter() -> Iterator over (NAME, TAG, CERT);
```

5.5. Input/Output Types

5.5.1. NAME

A string representing a fingerprint or a special name (see Section 3.2).

5.5.2. TAG

An opaque value corresponding to a cert in store. If the cert is updated, its tag will change. This can be used to quickly determine if an index data structure must be updated.

5.5.3. CERT

Exactly one OpenPGP certificate (section 11.1 of [RFC4880]), aka "Transferable Public Key". The certificate **MUST NOT** be ASCII Armored.

5.5.4. KEY

Exactly one OpenPGP Transferable Secret Key (section 11.2 of [RFC4880]). The certificate **MUST NOT** be ASCII Armored.

5.6. Failure Modes

Mnemonic	Meaning
OK	Success
BAD_NAME	The name was neither a valid fingerprint, nor a known special name
NOT_A_STORE	The base directory cannot possibly contain a store
BAD_DATA	The data was not valid OpenPGP cert or key in binary format
IO_ERROR	Unspecified I/O error occurred

Table 3

6. Guidance for Implementers

Despite the fact that this spec is designed with ease of implementation in mind, and we explicitly invite reimplementations, please consider using our reference implementation.

This is a list of implementation considerations that interoperating implementations need to follow:

- The exclusive lock **MUST** be released in a timely manner.

- When exporting artifacts from the store, non-exportable signatures and certificate components **MUST** be omitted.

7. Security Considerations

XXX

8. Document Considerations

8.1. Document History

This is a first draft that has not been published.

8.2. Future Work

OpenPGP requires efficient lookup by subkey fingerprint and keyids. This is currently not provided by this spec, hence implementations need to build their own index on top of this store. Future revisions may specify a way to do this natively.

Collecting usage information for TOFU-like trust models creates a write-heavy workload during normal usage, and requires more complex data structures that are not easily expressed using file-system operations and OpenPGP data structures. Future revisions of this spec may define suitable mechanisms to keep a record of certificate uses.

This spec contains platform-specific conventions (see Section 3.8), like default store locations and locking mechanisms. Porting to new platforms requires amending the spec.

9. Acknowledgements

10. References

10.1. Normative References

[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/info/rfc2119>.

[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/info/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/info/rfc8174>.

10.2. Informative References

[Autocrypt] "Convenient End-to-End Encryption for E-Mail", 14 June 2021, <https://autocrypt.org/>.

[FedoraSharedX.509CertStore] "Shared System Certificates", 14 June 2021, <https://fedoraproject.org/wiki/Features/SharedSystemCertificates>.

[I-D.draft-bre-openpgp-samples-00] Einarsson, B. R., "juga", and D. K. Gillmor, "OpenPGP Example Keys and Certificates", Work in Progress, Internet-Draft, draft-bre-openpgp-samples-00, 15 October 2019, <https://www.ietf.org/archive/id/draft-bre-openpgp-samples-00.txt>.

[I-D.draft-koch-openpgp-webkey-service-12] Koch, W., "OpenPGP Web Key Directory", Work in Progress, Internet-Draft, draft-koch-openpgp-webkey-service-12, 17 May 2021, <https://www.ietf.org/archive/id/draft-koch-openpgp-webkey-service-12.txt>.

[keys.openpgp.org] "A GDPR-conforming, validating keyserver", 14 June 2021, <https://keys.openpgp.org/>.

[macOSSharedX.509CertStore] "Lists of available trusted root certificates in macOS", 8 December 2018, <https://support.apple.com/en-us/HT202858>.

[Maildir] "Using maildir format", 14 June 2021, <https://cr.yp.to/proto/maildir.html>.

[Notmuch] "Notmuch -- Just an email system", 14 June 2021, <https://notmuchmail.org/>.

[reference-implementation-api] "API documentation for the reference implementation", 15 June 2021, <https://sequoia-pgp.gitlab.io/pgp-cert-d/pgp_cert_d/index.html>.

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

[WindowsSharedX.509CertStore] "Managing Certificates with Certificate Stores", 14 June 2021, <https://docs.microsoft.com/en-us/windows/win32/seccrypto/managing-certificates-with-certificate-stores>.

[X509-PKI] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[Zookos-Triangle] "Names: Distributed, Secure, Human-Readable: Choose Two", 17 June 2021, <https://web.archive.org/web/20011020191610/http://zooko.com/distnames.html>.

Authors' Addresses

Nora Widdecke
Sequoia PGP

Email: nora@sequoia-pgp.org

Justus Winter
Sequoia PGP

Email: justus@sequoia-pgp.org