Workgroup: openpgp
Internet-Draft: draft-dkg-openpgp-stateless-cli-02
Published: 29 October 2019
Intended Status: Informational
Expires: 1 May 2020
Author: D.K. Gillmor
ACLU

Stateless OpenPGP Command Line Interface

Abstract

This document defines a generic stateless command-line interface for dealing with OpenPGP messages, known as sop. It aims for a minimal, well-structured API covering OpenPGP object security.

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 1 May 2020.

Copyright Notice

Table of Contents

1. Introduction

Different OpenPGP implementations have many different requirements, which typically break down in two main categories: key/certificate management and object security.

The purpose of this document is to provide a "stateless" interface that primarily handles the object security side of things, and assumes that secret key management and certificate management will be handled some other way.

This separation should make it easier to provide interoperability testing for the object security work, and to allow implementations to consume and produce new cryptographic primitives as needed.

This document defines a generic stateless command-line interface for dealing with OpenPGP messages, known here by the placeholder `sop`. It aims for a minimal, well-structured API.

An OpenPGP implementation should not name its executable `sop` to implement this specification, of course. It just needs to provide a binary that conforms to this interface.

A `sop` implementation should leave no trace on the system, and its behavior should not be affected by anything other than command-line arguments and input.

Obviously, the user will need to manage their secret keys (and their peers' certificates) somehow, but the goal of this interface is to separate out that task from the task of interacting with OpenPGP messages.

While this document identifies a command-line interface, the rough outlines of this interface should also be amenable to relatively straightforward library implementations in different languages.

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" to refer to OpenPGP Transferable Public Key (see section 11.1 of [RFC4880]).

"Stateless" in "Stateless OpenPGP" means avoiding secret key and certificate state. The user is responsible for managing all OpenPGP certificates and secret keys themselves, and passing them to sop as needed. The user should also not be concerned that any state could affect the underlying operations.

OpenPGP revocations can have "Reason for Revocation" (section 5.2.3.23 of [RFC4880]), which can be either "soft" or "hard". The set of "soft" reasons is: "Key is superseded" and "Key is retired and no longer used". All other reasons (and revocations that do not state a reason) are "hard" revocations.

2. Examples

These examples show no error checking, but give a flavor of how `sop` might be used in practice from a shell.

The key and certificate files described in them (e.g. `alice.sec`) could be for example those found in [I-D.draft-bre-openpgp-samples-00].

```
sop generate-key "Alice Lovelace <alice@openpgp.example>" > alice.sec
sop extract-cert < alice.sec > alice.pgp

sop sign --as=text alice.sec < announcement.txt >
announcement.txt.asc
sop verify announcement.txt.asc alice.pgp < announcement.txt

sop encrypt --sign-with=alice.sec --as=mime bob.pgp < msg.eml >
encrypted.asc
sop decrypt alice.sec < ciphertext.asc > cleartext.out
```

confusing

make arguments options so they are explicit and not order dependent

3. Subcommands

`sop` uses a subcommand interface, similar to those popularized by systems like `git` and `svn`.

If the user supplies a subcommand that `sop` does not implement, it fails with a return code of 69. If a `sop` implementation does not handle a supplied option for a given subcommand, it fails with a return code of 37.

For all commands that have an `--armor|--no-armor` option, it defaults to `--armor`, meaning that any output OpenPGP material should be ASCII-armored (section 6 of [I-D.ietf-openpgp-rfc4880bis]) by default.

3.1. version: Version Information

```
sop version
```

- Standard Input: ignored
- Standard Output: version string

The version string emitted should contain the name of the `sop` implementation, followed by a single space, followed by the version number.

Example:

```
$ sop version
ExampleSop 0.2.1
$
```

3.2. generate-key: Generate a Secret Key

```
sop generate-key [--armor|--no-armor] [--] [USERID…]
```

- Standard Input: ignored
- Standard Output: `KEY` (Section 5.2)

Generate a single default OpenPGP certificate with zero or more User IDs.

Example:

```
$ sop generate-key 'Alice Lovelace <alice@openpgp.example>' >
alice.sec
$ head -n1 < alice.sec
-----BEGIN PGP PRIVATE KEY BLOCK-----
$
```

3.3. extract-cert: Extract a Certificate from a Secret Key

```
sop extract-cert [--armor|--no-armor]
```

- Standard Input: `KEY` (Section 5.2)
- Standard Output: `CERTS` (Section 5.1)

Note that the resultant `CERTS` object will only ever contain one OpenPGP certificate.

Example:

```
$ sop extract-cert < alice.sec > alice.pgp
$ head -n1 < alice.pgp
-----BEGIN PGP PUBLIC KEY BLOCK-----
$
```

3.4. sign: Create a Detached Signature

```
sop sign [--armor|--no-armor]
     [--as={binary|text}] [--] KEY [KEY...]
```

- Standard Input: `DATA` (Section 5.8)
- Standard Output: `SIGNATURE` (Section 5.4)

`--as` defaults to `binary`. If `--as=text` and the input `DATA` is not valid `UTF-8`, `sop sign` fails with a return code of 53.

Example:

```
$ sop sign --as=text alice.sec < message.txt > message.txt.asc
$ head -n1 < message.txt.asc
-----BEGIN PGP SIGNATURE-----
$
```

3.5. verify: Verify a Detached Signature

```
sop verify [--not-before=DATE] [--not-after=DATE]
    [--] SIGNATURE CERTS [CERTS...]
```

- Standard Input: `DATA` (Section 5.8)
- Standard Output: `VERIFICATIONS` (Section 5.7)

`--not-before` and `--not-after` indicate that signatures with dates outside certain range MUST NOT be considered valid.

`--not-before` defaults to the beginning of time. Accepts the special value `-` to indicate the beginning of time (i.e. no lower boundary).

`--not-after` defaults to the current system time (`now`). Accepts the special value `-` to indicate the end of time (i.e. no upper boundary).

`sop verify` only returns 0 if at least one certificate included in any `CERTS` object made a valid signature in the range over the `DATA` supplied.

For details about the valid signatures, the user MUST inspect the `VERIFICATIONS` output.

If no `CERTS` are supplied, `sop verify` fails with a return code of 19.

If no valid signatures are found, `sop verify` fails with a return code of 3.

See Section 9.1 for more details about signature verification.

Example:

(In this example, we see signature verification succeed first, and then fail on a modified version of the message.)

```
$ sop verify message.txt.asc alice.pgp < message.txt
2019-10-29T18:36:45Z EB85BB5FA33A75E15E944E63F231550C4F47E38E
EB85BB5FA33A75E15E944E63F231550C4F47E38E signed by alice.pgp
$ echo $?
0
$ tr a-z A-Z < message.txt | sop verify message.txt.asc alice.pgp
$ echo $?
3
$
```

3.6. encrypt: Encrypt a Message

```
sop encrypt [--as={binary|text|mime}]
    [--armor|--no-armor]
    [--with-password=PASSWORD...]
    [--sign-with=KEY...]
    [--] [CERTS...]
```

- Standard Input: `DATA` (Section 5.8)
- Standard Output: `CIPHERTEXT` (Section 5.3)

`--as` defaults to `binary`.

`--with-password` enables symmetric encryption (and can be used multiple times if multiple passwords are desired). If `sop encrypt` encounters a `PASSWORD` which is not a valid `UTF-8` string, it fails with a return code of 31. If `sop encrypt` sees trailing whitespace at the end of a `PASSWORD`, it will trim the trailing whitespace before using the password.

`--sign-with` enables signing by a secret key (and can be used multiple times if multiple signatures are desired).

If `--as` is set to either `text` or `mime`, then `--sign-with` will sign as a canonical text document. In this case, if the input `DATA` is not valid `UTF-8`, `sop encrypt` fails with a return code of 53.

The resulting `CIPHERTEXT` should be decryptable by the secret keys corresponding to every certificate included in all `CERTS`, as well as each password given with `--with-password`.

If no `CERTS` or `--with-password` options are present, `sop encrypt` fails with a return code of 19.

If at least one of the identified certificates requires encryption to an unsupported asymmetric algorithm, `sop encrypt` fails with a return code of 13.

If at least one of the identified certificates is not encryption-capable (e.g., revoked, expired, no encryption-capable flags on primary key and valid subkeys), `sop encrypt` fails with a return code of 17.

If `sop encrypt` fails for any reason, it emits no `CIPHERTEXT`.

Example:

(In this example, `bob.bin` is a file containing Bob's binary-formatted OpenPGP certificate. Alice is encrypting a message to both herself and Bob.)

```
$ sop encrypt --as=mime --sign-with=alice.key alice.asc bob.bin <
message.eml > encrypted.asc
$ head -n1 encrypted.asc
-----BEGIN PGP MESSAGE-----
$
```

3.7. decrypt: Decrypt a Message

```
sop decrypt [--session-key-out=SESSIONKEY]
    [--with-session-key=SESSIONKEY...]
    [--with-password=PASSWORD...]
    [--verify-out=VERIFICATIONS
     [--verify-with=CERTS...]
     [--verify-not-before=DATE]
     [--verify-not-after=DATE] ]
    [--] [KEY...]
```

- Standard Input: `CIPHERTEXT` (Section 5.3)
- Standard Output: `DATA` (Section 5.8)

`--session-key-out` can be used to learn the session key on successful decryption.

If `sop decrypt` fails for any reason and the identified `--session-key-out` file already exists in the filesystem, the file will be unlinked.

`--with-session-key` enables decryption of the `CIPHERTEXT` using the session key directly against the `SEIPD` packet. This option can be used multiple times if several possible session keys should be tried.

`--with-password` enables decryption based on any `SKESK` packets in the `CIPHERTEXT`. This option can be used multiple times if the user wants to try more than one password.

If `sop decrypt` tries and fails to use a supplied `PASSWORD`, and it observes that there is trailing `UTF-8` whitespace at the end of the `PASSWORD`, it will retry with the trailing whitespace stripped.

`--verify-out` produces signature verification status to the designated file.

`sop decrypt` does not fail (that is, the return code is not modified) based on the results of signature verification. The caller MUST check the returned `VERIFICATIONS` to confirm signature status. An empty `VERIFICATIONS` output indicates that no valid signatures were found. If `sop decrypt` itself fails for any reason, and the identified `VERIFICATIONS` file already exists in the filesystem, the file will be unlinked.

`--verify-with` identifies a set of certificates whose signatures would be acceptable for signatures over this message.

If the caller is interested in signature verification, both `--verify-out` and at least one `--verify-with` must be supplied. If only one of these arguments is supplied, `sop decrypt` fails with a return code of 23.

`--verify-not-before` and `--verify-not-after` provide a date range for acceptable signatures, by analogy with the options for `sop verify` (see Section 3.5). They should only be supplied when doing signature verification.

See Section 9.1 for more details about signature verification.

If no `KEY` or `--with-password` or `--with-session-key` options are present, `sop decrypt` fails with a return code of 19.

If unable to decrypt, `sop decrypt` fails with a return code of 29.

`sop decrypt` only returns cleartext to Standard Output that was successfully decrypted.

Example:

(In this example, Alice stashes and re-uses the session key of an encrypted message.)

```
$ sop decrypt --session-key-out=session.key alice.sec <
ciphertext.asc > cleartext.out
$ ls -l ciphertext.asc cleartext.out
-rw-r--r-- 1 user user   321 Oct 28 01:34 ciphertext.asc
-rw-r--r-- 1 user user   285 Oct 28 01:34 cleartext.out
$ sop decrypt --with-session-key=session.key < ciphertext.asc >
cleartext2.out
$ diff cleartext.out cleartext2.out
$
```

3.8. armor: Add ASCII Armor

```
sop armor [--label={auto|sig|key|cert|message}]
    [--allow-nested]
```

- Standard Input: 8-bit, unarmored OpenPGP material (`SIGNATURE`, `CERTS`, `KEY`, or `CIPHERTEXT`)
- Standard Output: the same material with ASCII-armoring added

The user can choose to specify the label used in the header and tail of the armoring. The default is `auto`, in which case, `sop` inspects the input and chooses the label appropriately. In this case, if `sop` cannot select a label on the basis of the input, it treats it as literal data, and labels it as a `message`.

If the incoming data is already armored, and the `--allow-nested` flag is not specified, the data MUST be output with no modifications. Data is considered ASCII armored iff the first 14 bytes are exactly `-----BEGIN PGP`. This operation is thus idempotent by default.

Example:

```
$ sop armor < bob.bin > bob.pgp
$ head -n1 bob.pgp
-----BEGIN PGP PUBLIC KEY BLOCK-----
$
```

3.9. dearmor: Remove ASCII Armor

```
sop dearmor
```

- Standard Input: ASCII-armored OpenPGP material (`CIPHERTEXT`, `SIGNATURE`, `CERTS`, or `KEY`)
- Standard Output: the same material with ASCII-armoring removed

Example:

```
$ sop dearmor < message.txt.asc > message.txt.sig
$
```

4. Input String Types

Some material is passed to `sop` directly as a string on the command line.

4.1. DATE

An ISO-8601 formatted timestamp with time zone, or the special value `now` to indicate the current system time.

Examples:

```
now
2019-10-29T12:11:04+00:00
2019-10-24T23:48:29Z
20191029T121104Z
```

In some cases where used to specify lower and upper boundaries, a `DATE` value can be set to `-` to indicate "no time limit".

A flexible implementation of `sop` MAY accept date inputs in other unambiguous forms.

4.2. USERID

This is an arbitrary `UTF-8` string. By convention, most User IDs are of the form `Display Name <email.address@example.com>`, but they do not need to be.

5. Input/Output Indirect Types

Some material is passed to `sop` indirectly, typically by referring to a filename containing the data in question. This type of data may also be passed to `sop` on Standard Input, or delivered by `sop` to Standard Output.

If the filename for any indirect material used as input has the special form `@ENV:xxx`, then contents of environment variable `$xxx` is used instead of looking in the filesystem.

If the filename for any indirect material used as either input or output has the special form `@FD:nnn` where `nnn` is a decimal integer, then the associated data is read from file descriptor `nnn`.

If any input data does not meet the requirements described below, `sop` will fail with a return code of 41.

5.1. CERTS

One or more OpenPGP certificates (section 11.1 of [I-D.ietf-openpgp-rfc4880bis]), aka "Transferable Public Key". May be armored.

Although some existing workflows may prefer to use one `CERTS` object with multiple certificates in it (a "keyring"), supplying exactly one certificate per `CERTS` input will make error reporting clearer and easier.

5.2. KEY

Exactly one OpenPGP Transferable Secret Key (section 11.2 of [I-D.ietf-openpgp-rfc4880bis]). May be armored.

Secret key material should be in cleartext (that is, it should not be locked with a password). If the secret key material is locked with a password, `sop` may fail to use the key.

5.3. CIPHERTEXT

`sop` accepts only a restricted subset of the arbitrarily-nested grammar allowed by the OpenPGP Messages definition (section 11.3 of [I-D.ietf-openpgp-rfc4880bis]).

In particular, it accepts and generates only:

An OpenPGP message, consisting of a sequence of PKESKs (section 5.1 of [I-D.ietf-openpgp-rfc4880bis]) and SKESKs (section 5.3 of [I-D.ietf-openpgp-rfc4880bis]), followed by one SEIPD (section 5.14 of [I-D.ietf-openpgp-rfc4880bis]).

The SEIPD can decrypt into one of two things:

- "Maybe Signed Data" (see below), or
- Compressed data packet that contains "Maybe Signed Data"

"Maybe Signed Data" is a sequence of:

- N (zero or more) one-pass signature packets, followed by
- zero or more signature packets, followed by
- one Literal data packet, followed by
- N signature packets (corresponding to the outer one-pass signatures packets)

FIXME: does any tool do compression inside signing? Do we need to handle that?

May be armored.

5.4. SIGNATURE

One or more OpenPGP Signature packets. May be armored.

5.5. SESSIONKEY

This documentation uses the GnuPG defacto `ASCII` representation:

`ALGONUM:HEXKEY`

where `ALGONUM` is the decimal value associated with the OpenPGP Symmetric Key Algorithms (section 9.3 of [I-D.ietf-openpgp-rfc4880bis]).

Example AES-256 session key:

```
9:FCA4BEAF687F48059CACC14FB019125CD57392BAB7037C707835925CBF9F7BCD
```

5.6. PASSWORD

This is expected to be a `UTF-8` string, but for `sop decrypt`, any bytestring that the user supplies will be accepted. Note the details in `sop encrypt` and `sop decrypt` about trailing whitespace!

5.7. VERIFICATIONS

One line per successful signature verification. Each line has three structured fields delimited by a single space, followed by arbitrary text to the end of the line.

- ISO-8601 UTC datestamp
- Fingerprint of the signing key (may be a subkey)
- Fingerprint of primary key of signing certificate (if signed by primary key, same as the previous field)
- arbitrary text

Example:

```
2019-10-24T23:48:29Z C90E6D36200A1B922A1509E77618196529AE5FF8
C4BC2DDB38CCE96485EBE9C2F20691179038E5C6 certificate from dkg.asc
```

5.8. DATA

Cleartext, arbitrary data. This is either a bytestream or `UTF-8` text.

It MUST only be `UTF-8` text in the case of input supplied to `sop sign --as=text` or `sop encrypt --as={mime|text}`. If `sop` receives `DATA` containing non-`UTF-8` octets in this case, it will fail with return code 53.

6. Failure Modes

When `sop` succeeds, it will return 0 and emit nothing to Standard Error. When `sop` fails, it fails with a non-zero return code, and emits one or more warning messages on Standard Error. Known return codes include:

Return	Meaning
0	Success
3	No acceptable signatures found (`sop verify`)
13	Asymmetric algorithm unsupported (`sop encrypt`)

Return	Meaning
17	Certificate not encryption-capable (e.g., expired, revoked, unacceptable usage flags) (`sop encrypt`)
19	Missing required argument
23	Incomplete verification instructions (`sop decrypt`)
29	Unable to decrypt (`sop decrypt`)
31	Non-UTF-8 password (`sop encrypt`)
37	Unsupported option
41	Invalid data type (no secret key where `KEY` expected, etc)
53	Non-text input where text expected
69	Unsupported subcommand

Table 1

A `sop` implementation MAY return other error codes than those listed above.

7. Guidance for Implementers

`sop` uses a few assumptions that implementers might want to consider.

7.1. One OpenPGP Message at a Time

`sop` is intended to be a simple tool that operates on one OpenPGP object at a time. It should be composable, if you want to use it to deal with multiple OpenPGP objects.

FIXME: discuss what this means for streaming. The stdio interface doesn't necessarily imply streamed output.

7.2. Simplified Subset of OpenPGP Message

While the formal grammar for OpenPGP Message is arbitrarily nestable, `sop` constrains itself to what it sees as a single "layer" (see Section 5.3).

This is a deliberate choice, because it is what most consumers expect, and runaway recursion is bad news.

Note that an implementation of `sop decrypt` MAY choose to handle more complex structures, but if it does, it should document the other structures it handles and why it chooses to do so. We can use such documentation to improve future versions of this spec.

7.3. Validate Signatures Only from Known Signers

There are generally only a few signers who are relevant for a given OpenPGP message. When verifying signatures, `sop` expects that the caller can identify those relevant signers ahead of time.

7.4. Detached Signatures

`sop` deals with detached signatures as the baseline form of OpenPGP signatures.

The main problem this avoids is the trickiness of handling a signature that is mixed inline into the data that it is signing.

7.5. Reliance on Supplied Certs and Keys

A truly stateless implementation may find that it spends more time validating the internal consistency of certificates and keys than it does on the actual object security operations.

For performance reasons, an implementation may choose to ignore validation on certificate and key material supplied to it. The security implications of doing so depend on how the certs and keys are managed outside of `sop`.

8. Guidance for Consumers

While `sop` is originally conceived of as an interface for interoperability testing, it's conceivable that an application that uses OpenPGP for object security would want to use it.

FIXME: more guidance for how to use such a tool safely and efficiently goes here.

FIXME: if an encrypted OpenPGP message arrives without metadata, it is difficult to know which signers to consider when decrypting. How do we do this efficiently without invoking `sop decrypt` twice, once without `--verify-*` and again with the expected identity material?

9. Security Considerations

The OpenPGP object security model is typically used for confidentiality and authenticity purposes.

9.1. Signature Verification

In many contexts, an OpenPGP signature is verified to prove the origin and integrity of an underlying object.

When `sop` checks a signature (e.g. via `sop verify` or `sop decrypt --verify-with`), it MUST NOT consider it to be verified unless all of these conditions are met:

- The signature must be made by a signing-capable public key that is present in one of the supplied certificates
- The certificate and signing subkey must have been created before or at the signature time
- The certificate and signing subkey must not have been expired at the signature time
- The certificate and signing subkey must not be revoked with a "hard" revocation
- If the certificate or signing subkey is revoked with a "soft" revocation, then the signature time must predate the revocation
- The signing subkey must be properly bound to the primary key, and cross-signed
- The signature (and any dependent signature, such as the cross-sig or subkey binding signatures) must be made with strong cryptographic algorithms (e.g., not `MD5` or a 1024-bit `RSA` key)

Implementers MAY also consider other factors in addition to the origin and authenticity, including application-specific information.

For example, consider the application domain of checking software updates.

If software package Foo version 13.3.2 was signed on 2019-10-04, and the user receives a copy of Foo version 12.4.8 that was signed on 2019-10-16, it may be authentic and have a more recent signature date. But it is not an upgrade (12.4.8 < 13.3.2), and therefore it should not be applied automatically.

In such cases, it is critical that the application confirms that the other information verified is *also* protected by the relevant OpenPGP signature.

Signature validity is a complex topic, and this documentation cannot list all possible details.

9.2. Compression

The interface as currently specified does not allow for control of compression. Compressing and encrypting data that may contain both attacker-supplied material and sensitive material could leak information about the sensitive material (see the CRIME attack).

Unless an application knows for sure that no attacker-supplied material is present in the input, it should not compress during encryption.

10. Privacy Considerations

Material produced by `sop encrypt` may be placed on an untrusted machine (e.g., sent through the public `SMTP` network). That material may contain metadata that leaks associational information (e.g., recipient identifiers in PKESK packets). FIXME: document things like PURBs and `--hidden-recipient`)

10.1. Object Security vs. Transport Security

OpenPGP offers an object security model, but says little to nothing about how the secured objects get to the relevant parties.

When sending or receiving OpenPGP material, the implementer should consider what privacy leakage is implicit with the transport.

11. Document Considerations

[RFC Editor: please remove this section before publication]

This document is currently edited as markdown. Minor editorial changes can be suggested via merge requests at https://gitlab.com/dkg/openpgp-stateless-cli or by e-mail to the authors. Please direct all significant commentary to the public IETF OpenPGP mailing list: openpgp@ietf.org

11.1. Document History

substantive changes between -00 and -01:

- Changed `generate` subcommand to `generate-key`
- Changed `convert` subcommand to `extract-cert`
- Added "Input String Types" section as distinct from indirect I/O
- Made implicit arguments potentially explicit (e.g. `sop armor --label=auto`)
- Added `--allow-nested` to `sop armor` to make it idempotent by default
- Added fingerprint of signing (sub)key to `VERIFICATIONS` output
- Dropped `--mode` and `--session-key` arguments for `sop encrypt` (no plausible use, not needed for interop)
- Added `--with-session-key` argument to `sop decrypt` to allow for session-key-based decryption
- Added examples to each subcommand
- More detailed error codes for `sop encrypt`
- Move from `CERT` to `CERTS` (each `CERTS` argument might contain multiple certificates)

11.2. Future Work

- `detach-inband-signature-and-message` subcommand (split a clearsigned message into a message and a detached signature) (see Section 7.4)
- certificate transformation into popular publication forms:
  - WKD
  - DANE OPENPGPKEY
  - Autocrypt
- `sop encrypt` - specify compression? (see Section 9.2)
- `sop encrypt` - specify padding policy/mechanism?
- `sop decrypt` - how can it more safely handle zip bombs?
- `sop decrypt` - what should it do when encountering weakly-encrypted (or unencrypted) input?
- `sop encrypt` - minimize metadata (e.g. `--throw-keyids`)?

- handling secret keys that are locked with passwords?
- specify an error if a `DATE` arrives as input without a time zone?
- specify an error if a `sop` invocation sees multiple copies of a specific `@FD:n` input (e.g., `sop sign @FD:3 @FD:3`)
- add considerations about what it means for armored `CERTS` to contain multiple certificates - multiple armorings? one big blob?
- do we need an interface or option (for performance?) with the semantics that `sop` doesn't validate certificates internally, it just accepts whatever's given as legit data? (see Section 7.5)

12. Acknowledgements

This work was inspired by Justus Winter's [OpenPGP-Interoperability-Test-Suite].

The following people contributed helpful feedback and considerations to this draft, but are not responsible for its problems:

- Justus Winter
- Vincent Breitmoser
- Edwin Taylor
- Jameson Rollins
- Allan Nordhoey

References

Normative References

[I-D.ietf-openpgp-rfc4880bis] Koch, W., carlson, b., Tse, R., Atkins, D., and D. Gillmor, "OpenPGP Message Format", Internet-Draftdraft-ietf-openpgp-rfc4880bis-08, 6 September 2019 , <http://www.ietf.org/internet-drafts/draft-ietf-openpgp-rfc4880bis-08.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP14, RFC2119, DOI10.17487/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", RFC4880, DOI10.17487/RFC4880, November 2007 , <https://www.rfc-editor.org/info/rfc4880>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP14, RFC8174, DOI10.17487/RFC8174, May 2017 , <https://www.rfc-editor.org/info/rfc8174>.

Informative References

[I-D.draft-bre-openpgp-samples-00] Einarsson, B., juga, j., and D. Gillmor, "OpenPGP Example Keys and Certificates", Internet-Draftdraft-bre-openpgp-samples-00, 15 October 2019 , <http://www.ietf.org/internet-drafts/draft-bre-openpgp-samples-00.txt>.

[OpenPGP-Interoperability-Test-Suite] , "OpenPGP Interoperability Test Suite", 28 October 2019 , <https://tests.sequoia-pgp.org/>.

Author's Address

Daniel Kahn Gillmor
American Civil Liberties Union
125 Broad St.
New York, NY, 10004
United States
Email: dkg@fifthhorseman.net