Stateless OpenPGP Command Line Interface
draft-dkg-openpgp-stateless-cli-00

Abstract

This document defines a generic stateless command-line interface for
dealing with OpenPGP messages, known as "scp". It aims for a
minimal, well-structured API for dealing with 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 April 30, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(https://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.
# Table of Contents

1. Introduction ................................................. 3  
1.1. Requirements Language ................................. 3  
1.2. Terminology ............................................ 4  
2. Examples ................................................... 4  
3. Subcommands ............................................... 4  
3.1. Version Information .................................... 5  
3.2. Generate a Secret Key .................................. 5  
3.3. Convert a Secret Key to a Certificate ............... 5  
3.4. Create a Detached Signature ......................... 5  
3.5. Verify a Detached Signature ......................... 5  
3.6. Encrypt a Message ...................................... 6  
3.7. Decrypt a Message ...................................... 7  
3.8. Adding ASCII Armor .................................... 8  
3.9. Removing ASCII Armor .................................. 9  
4. Input/Output Indirect Types .............................. 9  
4.1. CERT ..................................................... 9  
4.2. KEY ..................................................... 9  
4.3. CIPHERTEXT ............................................. 9  
4.4. SIGNATURE ............................................ 10  
4.5. SESSIONKEY ........................................... 10  
4.6. PASSWORD ............................................... 11  
4.7. VERIFICATIONS ......................................... 11  
4.8. DATA ................................................... 11  
5. Failure modes .............................................. 11  
6. Guidance for Implementors ............................... 12  
6.1. One OpenPGP Message At a Time ..................... 12  
6.2. Simplified Subset of OpenPGP Message ............... 12  
6.3. Validate Signatures Only From Known Signers ...... 13  
6.4. Detached Signatures .................................. 13  
6.5. Reliance on Supplied Certs and Keys ............... 13  
7. Guidance for Consumers .................................. 13  
8. Security Considerations .................................. 14  
8.1. Signature Verification ............................... 14  
8.2. Compression ........................................... 15  
9. Privacy Considerations .................................... 15  
10. Document Considerations ................................. 15  
10.1. Document History ..................................... 16  
10.2. Future Work .......................................... 16  
11. Acknowledgements ........................................ 16  
12. References ............................................... 16  
12.1. Normative References ................................. 16  
12.2. Informative References .............................. 17  
Author’s Address ............................................. 17
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 an 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.

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 "Alice Lovelace <alice@openpgp.example>" > alice.sec
sop convert < 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

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 Information

sop version

  o Standard Input: ignored
  o 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.

3.2. Generate a Secret Key

sop generate [--armor|--no-armor] [--] [USERID...]

  o Standard Input: ignored
  o Standard Output: "KEY" (Section 4.2)

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

3.3. Convert a Secret Key to a Certificate

sop convert [--armor|--no-armor]

  o Standard Input: "KEY" (Section 4.2)
  o Standard Output: "CERT" (Section 4.1)

3.4. Create a Detached Signature

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

  o Standard Input: "DATA" (Section 4.8)
  o Standard Output: "SIGNATURE" (Section 4.4)

"--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.

3.5. Verify a Detached Signature

sop verify [--not-before=DATE] [--not-after=DATE] [--] SIGNATURE CERT [CERT...]
3.6. Encrypt a Message

sop encrypt [--as={binary|text|mime}]
[--armor|--no-armor]
[--mode={any|communications|storage}]
[--with-password=PASSWORD...]
[--session-key=SESSIONKEY]
[--sign-with=KEY...]
[++] [CERT...]

o Standard Input: "DATA" (Section 4.8)

o Standard Output: "CIPHERTEXT" (Section 4.3)

"--as" defaults to "binary".

"--mode" defaults to "any", meaning any encryption-capable subkey may be used.

"--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.

"--session-key" permits the encryptor to select the symmetric encryption algorithm and specific session key.

"--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 each identified "CERT".

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

3.7. Decrypt a Message

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

- Standard Input: "CIPHERTEXT" (Section 4.3)
- Standard Output: "DATA" (Section 4.8)

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

If "sop decrypt" fails for any reason and the identified "SESSIONKEY" file already exists in the filesystem, the file will be unlinked.

"--with-password" enables symmetric decryption (and can be used multiple times if the user wants to try more password are tried).

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 certificate 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". They should only be supplied when doing signature verification.

See Section 8.1 for more details about signature verification.

If no "KEY" or "--with-password" 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.

### 3.8. Adding ASCII Armor

sop armor [--label={sig|key|cert|message}]

- Standard Input: 8-bit, unarmored OpenPGP material ("SIGNATURE", "CERT", "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. If the user does not specify, "sop" inspects the input and chooses the label appropriately. If "sop" cannot select a label on the basis of the input, it treats it as literal data, and labels it as a "message".
3.9. Removing ASCII Armor

sop dearmor

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

4. 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.

4.1. CERT

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

4.2. KEY

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.

4.3. 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.

4.4. SIGNATURE

One or more OpenPGP Signature packets. May be armored.

4.5. 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]).

As input, "ALGONUM:" alone (with an empty "HEXKEY") means "user specifies the algorithm, but the implementation chooses an arbitrary key for the cipher."

Example AES256 session key:
4.6. 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!

4.7. VERIFICATIONS

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

- ISO-8601 UTC datestamp
- Fingerprint of primary key of signing certificate
- arbitrary text

Example:


4.8. 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 "encrypt --as=(mime|text)". If "sop" receives "DATA" containing non-"UTF-8" octets it will fail with return code 53.

5. 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")
- **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

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

### 6. Guidance for Implementors

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

#### 6.1. 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.

#### 6.2. 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 4.3).
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.

6.3. 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.

6.4. 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.

6.5. 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 are of doing so depend on how the certs and keys are managed outside of "sop".

7. 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?
8. Security Considerations

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

8.1. 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 should only consider it to be verified only if at least 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 "CERT"s
- The "CERT" and signing subkey must have been created before or at the signature time
- The "CERT" and signing subkey must not have been expired at the signature time
- The "CERT" and signing subkey must not be revoked with a "hard" revocation
- If the "CERT" 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 should typically 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.

### 8.2. 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 on the input, it should not compress during encryption.

### 9. Privacy Considerations

Material produced by "sop encrypt" may be placed on 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")

#### 9.1. 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.

### 10. 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
10.1. Document History

10.2. Future Work

- "split" subcommand (split a clearsigned message into a message and a detached signature) (see Section 6.4)

- certificate transformation into popular publication forms:
  * WKD
  * DANE OPENPGPKEY
  * Autocrypt

- "sop encrypt" - specify compression? (see Section 8.2)

- "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?

- do we need an interface (for performance?) that says "don’t validate certificates internally, just accept whatever’s given as legit data"? (see Section 6.5)

11. Acknowledgements

This work was inspired by Justus Winter’s [OpenPGP-Interoperability-Test-Suite], and discussions with Justus. Problems with this spec are not his fault.

12. References

12.1. Normative References

[I-D.ietf-openpgp-rfc4880bis]
12.2. Informative References


[I-D.draft-bre-openpgp-samples-00] Einarsson, B., juga, j., and D. Gillmor, "OpenPGP Example Keys and Certificates", draft-bre-openpgp-samples-00 (work in progress), October 2019.


Author’s Address

Daniel Kahn Gillmor
American Civil Liberties Union
125 Broad St.
New York, NY 10004
USA

Email: dkg@fifthhorseman.net