APERTIS

GPL-3-free replacements of GnuPG

Contents

Introduction 2

Terminology and concepts 3

Use cases 3

Non-use cases 3

Requirements 3

Depending components 4
- apertis-archive-keyring . . . 5
- APT . . . 5
- Flatpak . . . 5
- gmime . . . 5
- OSTree . . . 6
- volume-key . . . 6

Approach 6
- RNP . . . 6
- rPGP . . . 7
- Sequoia . . . 8
- golang.org/x/crypto/openpgp . . . 8
- gpgrv . . . 9

Evaluation Report 9
- Debian upstream discussion . . . 10

Recommendations 10

Risks 12

Introduction

In accordance to its Open Source License Expectations[1], Apertis currently ships a very old version of `GnuPG` which is still released under the `GPL-2.0` terms, before the upstream project switched to `GPL-3.0`.

This is problematic in the long term: the purpose of this document is to investigate alternative implementations with licensing conditions that are suitable for Apertis target devices.

The use cases for Apertis target images only depend on GnuPG for verification purposes, not for signing or encrypting. This is usually done through the `gpgv`

[1] https://www.apertis.org/policies/license-expectations/

tool or through the `libgpgme` library which invokes the `gpg` tool and interacts with it via the `--with-colons` machine parsable mode[2] or the Assuan[3] IPC protocol.

Newer `GPL-3`-licensed versions of GnuPG can be provided in the `development` package repository for any additional need outside that do not affect targets. Until `Ed25519` support is officially implemented in APT, the upstream version is imported in Apertis and our infrastructure is reworked to use it rather than OpenPGP signatures, we will need GnuPG to sign and install packages on development images. This does not affect production devices as APT is not meant to be used there.

Terminology and concepts

- **OpenPGP**: The OpenPGP protocol defines standard formats for encrypted messages, signatures, and certificates for exchanging public keys.
- **GnuPG**: GnuPG is a complete and free implementation of the OpenPGP standard.

Use cases

- A developer wants to install an additional package on the Apertis APT-based image flashed on their device, and relies on OpenPGP signatures to assert trust in the remote package repositories.
- A user wants to install a Flatpak application from Flathub, which only provides OpenPGP signatures to assert trust on the provided application bundles.

Non-use cases

- Sending emails encrypted with OpenPGP
- Creating OpenPGP signatures

Requirements

The chosen approach to replace GnuPG on targets must:

- have a license that matches the Apertis Open Source License Expectations[4], including its dependencies
- provide OpenPGP signature verification support
- require minimal changes in tools currently depending on GnuPG
- require minimal non-upstreamable changes

[2]https://github.com/gpg/gnupg/blob/master/doc/DETAILS
[3]https://www.gnupg.org/documentation/manuals/assuan/index.html
[4]https://www.apertis.org/policies/license-expectations/

- have an active upstream community
- have a high code quality track

Depending components

`GnuPG` and the related components are currently used in Apertis for the following packages (based on `apt-rdepends` results):

component	dependent package	source	repository
gnupg	flatpak-tests	flatpak	target
	libgpgme11	gpgme1.0	target
	libvolume-key1	volume-key	target
	ostree-tests	ostree	target
	python-apt		development
	devscripts		development
	gnupg2		development
	jetring		development
libgpgme11	flatpak	flatpak	target
	flatpak-tests	flatpak	target
	libflatpak0	flatpak	target
	gmime-bin	gmime	target
	libgmime-3.0-0	gmime	target
	libgpgmepp6	gpgme1.0	target
	libvolume-key1	volume-key	target
	samba-dsdb-modules	samba	development
gpgv	apertis-archive-keyring		target
	apt		target
	gnupg		target
	devscripts		development
	gpgv2		development

Current packages using `GnuPG` or `gpgv` are:

component	dependencies
apertis-archive-keyring	gpgv
apt	gpgv
flatpak	gnupg, libgpgme11
gmime	libgpgme11
ostree	gnupg, libgpgme11(1)
volume-key	gnupg, libgpgme11

(1) Currently `OSTree` in Apertis does not depend on `GnuPG` as it exclusively uses `Ed25519` signatures. However, the reintroduction of OpenPGP signature verifica-

tion support may be requested in the future to be able to verify the provenance and install applications from third-party Flatpak repositories that only provide OpenPGP signatures.

apertis-archive-keyring

This package contains all necessary GnuPG cryptographic keys needed to sign all Apertis archives. The runtime dependency on `gpgv` can be removed with no ill effect.

APT

`gpgv` is used by `APT`:

- to assert trust on remote package repository indexes
- by `apt-key` which is deprecated[5] and will be removed
- in build-time tests

Calls to `gpgv` are encapsulated in `ExecGPGV` function located in `apt-pkg/contrib/gpgv.cc`.

At the time this document is written, there's a discussion in Debian mailing list regarding ideas to replace gpgv with sqv[6]. The emerging long term idea is to have the `APT` code link to the Sequoia cryptographic library underlying `sqv`, rather than the current approach of invoking an external process.

Flatpak

Flatpak application and library use both `libgpgme11` and `libostree`.

`GnuPG` is used by `Flatpak`:

- during development to sign the package and summaries,
- and on target to verify the signatures.

Starting with Apertis v2022dev2, `Flatpak` is also able to use `Ed25519` cryptography.

gmime

`GnuPG` is used by `gmime` to encrypt, decrypt, sign and verify messages with `Multipurpose Internet Mail Extension`.

Starting with Apertis v2022dev3, the ability to encrypt, decrypt, sign and verify messages has been disabled in `gmime`.

[5] https://manpages.debian.org/testing/apt/apt-key.8.en.html
[6] https://lists.debian.org/deity/2021/01/msg00088.html

OSTree

`GnuPG` is used by `OSTree`:

- during development to sign the commits,
- and on target to verify the commits.

Current version of `OSTree` in Apertis is also able to use `Ed25519` cryptography.

volume-key

See Debian manpage[7].

`GnuPG` is used by `volume-key` to encrypt or decrypt the file used to store extracted "secrets"used for volume encryption (for example keys or passphrases).

Starting with Apertis v2022dev3, key escrow support has been disabled in `libblockdev` library, allowing to remove `volume-key`.

Approach

The following alternative replacements have been considered:

library	License	language	comment
RNP	BSD-2-Clause + BSD-3-Clause + Apache-2.0	C++	
rPGP	Apache-2.0 or MIT	Rust	
Sequoia	GPL-2+	Rust	uses Nettle/GMP
golang.org/x/crypto/openpgp	BSD-3-Clause	Golang	
gpgrv	Apache-2.0 or MIT	Rust	only provides gpg

RNP

https://github.com/rnpgp/rnp

Started in 2017.

RNP originated as an attempt to modernize the NetPGP codebase originally created by Alistair Crooks of NetBSD in 2016. RNP has been heavily rewritten, and carries minimal if any code from the original codebase

Version	# commits	# contributors	CI	gpgv replacement	C API
0.14	2700	31	yes	yes	yes

Used by:

[7]https://manpages.debian.org/buster/volume-key/volume_key.8.en.html

- Thunderbird
- EnMail[8] ruby gem

rPGP

https://github.com/rpgp/rpgp

Started in 2017.

Project description from rPGP site:

> rPGP is the only full Rust implementation of OpenPGP, following RFC4880 and RFC2440. It offers a minimal low-level API and does not prescribe trust schemes or key management policies. It fully supports all functionality required by the Autocrypt 1.1 e-mail encryption specification.
>
> …
>
> rPGP and its RSA dependency got a first independent security review mid 2019. No critical flaws were found. We have fixed and are fixing some high, medium and low risk ones. We will soon publish the full review report.
>
> Further independent security reviews are upcoming.
>
> …
>
> How is rPGP different from Sequoia?
>
> Some key differences:
>
> - rPGP has a more libre license than Sequoia that allows a broader usage
> - rPGP is a library with a well-defined, relatively small feature-set where Sequoia also tries to be a replacement for the GPG command line tool
> - All crypto used in rPGP is implemented in pure Rust, whereas sequoia uses Nettle, which is implemented in C.

Version	# commits	# contributors	CI	gpgv replacement	C API
0.7.1	334	12	no	no	no, but possible via a Rust shim

Used by:

- Delta Chat, the e-mail based messenger app suite[9]

[8]https://github.com/riboseinc/enmail
[9]https://delta.chat/

Sequoia

https://sequoia-pgp.org/

https://gitlab.com/sequoia-pgp/sequoia

Started in 2017.

Project status:

> The low-level API is quite feature-complete and can be used encrypt, decrypt, sign, and verify messages. It can create, inspect, and manipulate OpenPGP data on a very low-level.
>
> The high-level API is effectively non-existent, though there is some functionality related to key servers and key stores.
>
> The foreign function interface provides a C API for some of Sequoia' s low- and high-level interfaces, but it is incomplete.
>
> There is a mostly feature-complete command-line verification tool for detached messages called 'sqv'.

`Sequoia` uses Nettle[10] which is dual licensed LGPL-3.0 and GPL-2.0[11], see COPYING.LESSERv3, COPYINGv3, and COPYINGv2 files in the Nettle source repository[12]. This is compliant with the Apertis Open Source License Expectations[13] since Sequoia itself is licensed under the GPL-2.0 terms.

Version	# commits	# contributors	CI	gpgv replacement	C API
library: 1.0.0 Command line tools: 0.23.0	3948	33	yes	yes	yes

Used by:

- Pijul, KIPA, Radicle, see https://sequoia-pgp.org/projects/

`Sequoia` is already packaged for Debian bullseye.

golang.org/x/crypto/openpgp

https://pkg.go.dev/golang.org/x/crypto/openpgp

[10] https://git.lysator.liu.se/nettle/nettle
[11] http://www.lysator.liu.se/~nisse/nettle/nettle.html#Copyright
[12] https://git.lysator.liu.se/nettle/nettle
[13] https://www.apertis.org/policies/license-expectations/

https://github.com/golang/crypto/tree/master/openpgp

This package is part of the Go crypto package.

Version	# commits	# contributors	CI	gpgv replacement	C API
v0.0.0-20201221181555-eec23a3978ad			no	no	no

Used by:

- Imported by a lot of Go projects, see https://pkg.go.dev/golang.org/x/crypto/openpgp?tab=importedby

gpgrv

https://github.com/FauxFaux/gpgrv

Started in 2017.

`gpgrv` is a Rust library for verifying some types of GPG signatures.

It currently able to verify RSA, SHA1, SHA256 and SHA512 signatures.

Version	# commits	# contributors	CI	gpgv replacement	C API
0.3.0[14]	109	2	no	yes	NA

Used by:

- APT

Evaluation Report

The `golang.org/x/crypto/openpgp` package only provides a Go interface and would then require substantial effort to be integrated in other places.

`gpgrv` doesn't seem to be actively developed, with the last commit being on August 2020.

`RNP` and `Sequoia` provide C interfaces and CLI interfaces to encrypt, decrypt, sign or verify files. They have both received a lot of commits, and have many contributors.

`rPGP` does not provide any CLI interface and a C interface would require a Rust shim, but its licensing terms are much more flexible than the Sequoia ones. It is actively developed. but it has fewer commits and contributors than Sequoia.

[14] https://crates.io/crates/gpgrv

Red Hat removed the OpenPGP support from Thunderbird in Red Hat Enterprise Linux (RHEL), which uses `RNP`, due to not wanting to distribute Botan[15], which has inadequate side-channel protection, see Red Hat bugs 1837512[16] and 1886958[17].

Debian upstream discussion

The Debian APT maintainers are discussing and planning the removal of the dependency on `gpgv` and potentially on OpenPGP as a whole.

For the replacement of `gpgv` Debian will likely not use `RNP` due to its Apache License, see here[18], and expressed some interest in linking directly to Sequoia[19].

However, the Debian APT maintainers expressed concrete interest in moving away from OpenPGP altogether[20], by changing the signature mechanism to use Ed25519 instead[21].

Adopting a solution which is aligned to the upstream goals would save maintenance effort in the long term.

Recommendations

The split between `rPGP` (more permissive license, more limited goals) and Sequoia (more active, GPL-2.0 only) is unfortunate since `rPGP` would be more suitable for us but is also more risky regarding long term maintenance, with Sequoia being more promising in this regard.

The problems to be addressed are:

1. the use of GnuPG via `gpgv` on the target reference images
2. the use of GnuPG via `libgpgme` on the target reference images

For `gpgv` there are two possible approaches:

1. use `sqv` from Sequoia to replace `gpgv` with basically no changes in the depending components
2. for GPL-2.0 applications, link to Sequoia directly as the APT maintainers said

For `libgpgme` the situation is more complex because the API surface is way bigger and there are no drop-in replacements. In addition Sequoia, by being GPL-2.0 licensed, is not suitable to be directly linked from `GMime`, `OSTree` and `Flatpak` which are LGPL-2.1 and provide libraries that are meant to be linked by

[15] https://botan.randombit.net/
[16] https://bugzilla.redhat.com/show_bug.cgi?id=1837512
[17] https://bugzilla.redhat.com/show_bug.cgi?id=1886958
[18] https://lists.debian.org/deity/2021/02/msg00011.html
[19] https://lists.debian.org/deity/2021/02/msg00004.html
[20] https://lists.debian.org/deity/2021/02/msg00023.html
[21] https://wiki.debian.org/Teams/Apt/Spec/AptSign

applications that may be released under licenses incompatible with the GPL-2.0 or even proprietary. `rPGP` may be a better choice in this regard.

The approach could then be:

1. ship `sqv` on target images and create a new `sequoia-gpgv` wrapper which sends the correct status codes so that it gets transparently picked up by APT
2. patch `apertis-archive-keyring` to avoid any runtime dependency on GnuPG
3. disable OpenPGP support from `OSTree`, replacing it with the use of Ed25519 signatures
   - this will drop the ability to assert trust when pulling from third party OpenPGP-signed repositories, which has never been a use-case of interest in Apertis
4. disable OpenPGP support from `Flatpak`, replacing it with the use of Ed25519 signatures
   - this will drop the ability to assert trust when pulling from third party Flatpak repositories, which is not a use-case of interest for Apertis target devices but at some point is likely to be desirable on the SDK, so we may consider re-introducing GnuPG support only there where the GPL-3 restrictions are not a concern
5. disable OpenPGP support from `GMime`
   - this will drop the ability to send/receive encrypted emails when using evolution-data-server, which has never been a use-case of interest in Apertis
6. disable key escrow support from `libblockdev` so we can drop the `volume-key` package as a whole with its dependency on `libgpgme`
7. move the `gpgme` source package to the `development` package repository
8. move the `gnupg` source package to the `development` package repository
9. re-align the `gnupg` source package to Debian

With the steps above it would be possible to stop shipping an outdated GnuPG version with limited effort and limited regressions. In particular, disabling OpenPGP support from Flatpak means that it would not be possible to verify the provenance of applications shipped by third-party stores which use OpenPGP like Flathub, and disabling it from GMime would mean that it could not verify or decrypt OpenPGP emails: both regressions have a very limited impact on the Apertis use-cases.

In the longer term, other activities can be undertaken to get rid of the downstream delta introduced above:

1. engage with the APT upstream maintainers to help them move away from OpenPGP signatures[22]
2. engage with OSTree and Flatpak upstream maintainers to dynamically load `libgpgme` that it can be picked up on the SDK where installing GPL-3.0

[22]https://wiki.debian.org/Teams/Apt/Spec/AptSign

components is not an issue and where it can be useful to install applications from third-party store like Flathub

3. engage with Flathub people to support `Ed25519` signatures in addition to the OpenPGP ones
4. fully re-enable OpenPGP support in the components where it has been disabled by either:
5. porting them to use `rPGP` by engaging with the upstream maintainers about implementing minimal Rush shims
6. implementing a `libgpgme` backend that invokes Sequoia externally to avoid licensing issues, either by engaging with the `libgpgme` maintainers or the Sequoia maintainers by providing compatibility with the `--with-colons` machine parsable mode[23]

Risks

Drop-in reimplementations may not be 100% compatible and thus may cause subtle issues.

[23]https://github.com/gpg/gnupg/blob/master/doc/DETAILS