Post-quantum OpenPGP with Evolution on Fedora 45 GNOME

Stavros Kousidis

Published 2026-09-28
Tested on Fedora 45 GNOME · Evolution 3.62.0 · Evolution Data Server 3.62.0 · GPGME 2.0.1-6.fc45 (stock Fedora) · Sequoia Chameleon GnuPG 0.13.1 · sequoia-openpgp 2.4.1 · sequoia-sq 1.4.1

This guide covers the Evolution-specific part of the Fedora 45 RFC 9980 proof of concept. Complete the shared setup first.

Evolution is architecturally different from KMail and Claws Mail. Its OpenPGP path uses Camel's GnuPG-compatible command-line integration instead of the GPGME path exercised by the other two clients:

```
Evolution
  -> Camel OpenPGP
  -> GnuPG-compatible CLI
  -> Sequoia Chameleon
  -> Sequoia OpenPGP / Keystore
```

The runtime checks below confirm this distinction.

Install Evolution and verify the environment

Install Evolution:

```
sudo dnf install -y evolution
```

The shared setup restores Fedora's stock GPGME for the Evolution test. Verify the installed Evolution packages and confirm that GPGME is the stock Fedora build:

```
rpm -q evolution evolution-data-server gpgme
```

The tested baseline was:

```
Evolution 3.62.0
Evolution Data Server 3.62.0
gpgme-2.0.1-6.fc45.x86_64
```

Evolution's successful RFC 9980 OpenPGP path does not require the experimental GPGME algorithm patch.

Start Evolution in the Chameleon environment

Use the same terminal in which you set the compatibility `PATH` in the shared setup. Close any existing Evolution instance before continuing.

Confirm the selected OpenPGP command:

```
command -v gpg
gpgconf --list-components | grep -E '^(gpg:|gpgsm:)'
```

Start Evolution once from this terminal and keep this instance open for the remaining steps:

```
evolution >/tmp/evolution-rfc9980.log 2>&1 &
EVOLUTION_PID=$!
sleep 8
kill -0 "$EVOLUTION_PID"
```

If the last command fails, inspect `/tmp/evolution-rfc9980.log` and check whether an earlier Evolution instance was still running.

Inspect whether the main process maps Camel and GPGME:

```
grep -E 'libcamel|libgpgme' "/proc/$EVOLUTION_PID/maps" \
  | awk '{print $6}' | sort -u
```

The tested main process loaded `libcamel-1.2.so.68.0.0` but did not map `libgpgme`.

Runtime proof of the Camel CLI path

The main Evolution process loaded:

```
/usr/lib64/libcamel-1.2.so.68.0.0
/usr/lib64/libgpg-error.so.0.42.1
```

but not `libgpgme`.

During the original interoperability test, a temporary tracing wrapper in front of the Chameleon `gpg` executable recorded the commands spawned by Evolution. This tracing step is evidence from that test rather than a requirement for completing the walkthrough.

For signing, the captured invocation contained:

```
--sign --detach --armor -u rfc9980-poc@example.invalid --output -
```

For public-key export, Camel invoked:

```
--export
--export-options export-minimal,no-export-attributes
--export-filter ...
<rfc9980-poc@example.invalid>
```

This is direct runtime evidence for:

```
Evolution
  -> Camel
  -> gpg-compatible CLI
  -> Chameleon
```

rather than a GPGME-based OpenPGP path.

Configure the test account

The first-run assistant creates the test account. Use:

Identity:

```
Name:  RFC9980 PoC
Email: rfc9980-poc@example.invalid
```

Disable automatic server lookup.

Receiving:

Choose **Maildir-format mail directories**.

The tested walkthrough uses:

```
$HOME/Maildir
```

Create it if necessary:

```
mkdir -p "$HOME/Maildir"/{cur,new,tmp}
```

Sending:

```
SMTP server: localhost
Port: 25
Authentication: none
Encryption: none
```

The account name shown in Evolution's **Edit Accounts** dialogue may differ from the identity name. Rename it there if desired.

Keep Evolution's default OpenPGP key selection behavior:

```
OpenPGP Key ID:
  Use sender e-mail address

Always sign outgoing messages:
  disabled

Always encrypt outgoing messages:
  disabled

Always encrypt to myself when sending encrypted messages:
  enabled
```

With the `--userid "RFC9980 PoC <rfc9980-poc@example.invalid>"` certificate from the shared setup, no explicit fingerprint is required in the **OpenPGP Key ID** field.

Create and locate the three messages

Create three messages to:

```
rfc9980-poc@example.invalid
```

using:

```
Unsigned
Signed only
Encrypted + signed
```

In the compose window, enable the relevant OpenPGP controls through:

```
☰
 → Options
   → PGP Sign
   → PGP Encrypt
```

Create the three test messages as:

Subject	PGP Sign	PGP Encrypt
Unsigned	off	off
Signed only	on	off
Encrypted + signed	on	on

The receiving Maildir configured above and Evolution's local Outbox are separate stores. The test messages remain in Evolution's Outbox when the deliberately non-functional SMTP transport cannot send them.

The tested queued messages were stored below:

```
$HOME/.local/share/evolution/mail/local/.Outbox/cur/
```

Discover candidate Outbox directories instead of assuming the exact internal layout:

```
find "$HOME/.local/share/evolution/mail" -type d \
  \( -name '.Outbox' -o -name 'Outbox' \) -print
```

Choose the directory corresponding to Evolution's local Outbox and set `OUTBOX`, for example:

```
OUTBOX="$HOME/.local/share/evolution/mail/local/.Outbox"
```

The value above is the **tested layout**, not a portable constant. Inspect the message files:

```
find "$OUTBOX" -type f \( -path '*/cur/*' -o -path '*/new/*' \) \
  -printf '%T@ %p\n' | sort -nr | head
```

Match the messages by subject and MIME headers. Record the signed-only and encrypted-and-signed paths, then create a verification directory:

```
SIGNED="SIGNED_MESSAGE"
ENCRYPTED="ENCRYPTED_MESSAGE"
WORK="$HOME/rfc9980-mail-test/evolution-verify"
export SIGNED ENCRYPTED WORK

rm -rf "$WORK"
mkdir -p "$WORK"
```

Independent verification

Extract the signed-only PGP/MIME signature and the canonicalized signed MIME part:

```
python3 - <<'PY'
from email import policy
from email.parser import BytesParser
from pathlib import Path
import os

work = Path(os.environ["WORK"])
m = BytesParser(policy=policy.SMTP).parsebytes(Path(os.environ["SIGNED"]).read_bytes())
parts = list(m.iter_parts())
assert len(parts) >= 2, "Signed message does not have expected MIME parts"
work.joinpath("signed-content.txt").write_bytes(parts[0].as_bytes(policy=policy.SMTP))
work.joinpath("signed-signature.asc").write_bytes(parts[1].get_payload(decode=True))
PY

gpg --list-packets "$WORK/signed-signature.asc"

gpg --status-fd=1 \
  --verify "$WORK/signed-signature.asc" "$WORK/signed-content.txt"
```

Confirm an algorithm-`30` signature, `GOODSIG`, `VALIDSIG ... 30 ...`, and exit status `0`.

Extract and inspect the encrypted OpenPGP payload:

```
python3 - <<'PY'
from email import policy
from email.parser import BytesParser
from pathlib import Path
import os

work = Path(os.environ["WORK"])
m = BytesParser(policy=policy.SMTP).parsebytes(Path(os.environ["ENCRYPTED"]).read_bytes())
parts = list(m.iter_parts())
assert len(parts) >= 2, "Encrypted message does not have expected MIME parts"
work.joinpath("encrypted.pgp").write_bytes(parts[1].get_payload(decode=True))
PY

gpg --list-packets "$WORK/encrypted.pgp"

gpg --status-fd=1 \
  --output "$WORK/decrypted.eml" \
  --decrypt "$WORK/encrypted.pgp"
```

Confirm a version-6 public-key encrypted session key packet using algorithm `35` and `DECRYPTION_OKAY`. If Evolution's **Always encrypt to myself** option is enabled, the message can contain two identical algorithm-`35` PKESK packets.

Extract and verify the inner signature:

```
python3 - <<'PY'
from email import policy
from email.parser import BytesParser
from pathlib import Path
```

```
import os

work = Path(os.environ["WORK"])
m = BytesParser(policy=policy.SMTP).parsebytes(work.joinpath("decrypted.eml").read_bytes())
parts = list(m.iter_parts())
assert len(parts) >= 2, "Decrypted message is not multipart/signed"
work.joinpath("inner-content.txt").write_bytes(parts[0].as_bytes(policy=policy.SMTP))
work.joinpath("inner-signature.asc").write_bytes(parts[1].get_payload(decode=True))
PY

gpg --list-packets "$WORK/inner-signature.asc"

gpg --status-fd=1 \
  --verify "$WORK/inner-signature.asc" "$WORK/inner-content.txt"
```

Confirm another algorithm-`30` signature with `GOODSIG`, `VALIDSIG`, and exit status `0`.

Certificate identity interoperability

The shared setup uses a single explicit User ID:

```
RFC9980 PoC <rfc9980-poc@example.invalid>
```

and leaves Evolution on its default sender-e-mail-address key selection. This combination worked without requiring an explicit fingerprint in the **OpenPGP Key ID** field.

A certificate generated with separate `--name` and `--email` options also worked cryptographically with Evolution, but one test displayed:

```
Valid signature, but sender address and signer address do not match (RFC9980 PoC)
```

The certificate represented the identity as separate User IDs:

```
RFC9980 PoC
<rfc9980-poc@example.invalid>
```

Independent verification still returned `GOODSIG`, `VALIDSIG` and exit status `0`; the warning was an identity-matching issue rather than a signature failure. Adding another Combined User ID during that investigation did not remove the warning.

Both certificate-generation approaches are viable for the tested cryptographic operations. For this Evolution walkthrough, prefer the explicit `--userid` form containing the sender e-mail address because it gave the cleaner application-level result.

Additional GnuPG-CLI compatibility observation

Camel also uses:

```
--export-options export-minimal,no-export-attributes
```

A direct Chameleon 0.13.1 test rejected:

```
no-export-attributes
```

as an unknown export option.

This is a separate GnuPG-CLI compatibility gap worth tracking independently of the successful signing and encryption path.

Result

The procedure exercises this path:

```
Evolution
  -> Camel OpenPGP
  -> GnuPG-compatible Chameleon CLI
  -> Sequoia OpenPGP / Keystore
```

using Fedora's stock GPGME package.

Independent verification confirms:

```
Signing:
  ML-DSA-65+Ed25519
  OpenPGP algorithm 30
  GOODSIG / VALIDSIG

Encryption:
  ML-KEM-768+X25519
  version-6 PKESK
  OpenPGP algorithm 35
  DECRYPTION_OKAY

Inner signature:
  OpenPGP algorithm 30
  GOODSIG / VALIDSIG
```

The central architectural result is that Evolution does not require the experimental GPGME algorithm patch for this OpenPGP path.