Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2021

H. Marques
pEp Foundation

November 03, 2020

pretty Easy privacy (pEp): Email Formats and Protocols
draft-pep-email-01

Abstract

The proposed pretty Easy privacy (pEp) protocols for email are based upon already existing email and encryption formats (as PGP/MIME) and designed to allow for easily implementable and interoperable opportunistic encryption. The protocols range from key distribution, secret key synchronization between own devices, to mechanisms of metadata and content protection. The metadata and content protection is achieved by moving the whole message (not only the body part) into the PGP/MIME encrypted part. The proposed pEp Email Formats not only achieve simple forms of metadata protection (like subject encryption), but also allow for sending email messages through a mixnet. Such enhanced forms of metadata protection are explicitly discussed within the scope of this document.

The purpose of pEp for email is to simplify and automate operations in order to make usage of email encryption a viability for a wider range of Internet users, with the goal of achieving widespread implementation of data confidentiality and privacy practices in the real world.

The proposed operations and formats are targeted towards to Opportunistic Security scenarios and are already implemented in several applications of pretty Easy privacy (pEp).

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 May 7, 2021.

Table of Contents

1. Introduction

This document contains propositions for implementers of Mail User Agents (MUAs) seeking to support pretty Easy privacy (pEp) specifically for email [RFC5322]. All the propositions of [I-D.birk-pep] also apply to pEp for email. In this document, requirements are outlined for MUAs wanting to establish interoperability and/or to implement pEp for email.

pEp for email builds upon the cryptographic security services offered by PGP/MIME [RFC3156]. The primary goals of pEp for email are:

(1) Maximization of email privacy for Internet actors deploying and using the pretty Easy privacy approach.

(2) Compatibility with legacy or other automatic email encryption solutions in order to preserve privacy to the greatest extent possible.

Interoperability with S/MIME [RFC8551] is a also goal, but at this time there is no specification or running code that achieves this goal.

Current tools and implementations have failed to provide a sufficient level of usability to ordinary Internet users, with the result that end-to-end email encryption is seldom used.

While OpenPGP [RFC4880] using PGP/MIME [RFC3156] offers good encryption-for message contents at least-more work is needed to achieve the following three objectives of pretty Easy privacy (pEp):

1. make email encryption as automatic as possible,

2. protect as much metadata as possible, and

3. provide an easy way to authenticate communication partners.

A reference implementation of pEp for email is available for all major platforms and it has been ported to many programming languages (cf. Section 11 for an overview).

1.1. Relationship to other pEp documents

This document describes the pEp for email protocols. While it specifies details particularly related to pEp for email, it basically inherits the structure of [I-D.birk-pep], which describes the general concepts of pEp on a higher level.

For protocol details, constituent pEp mechanisms which also apply to email can be found in documents like [I-D.marques-pep-handshake]), which shows how trust between any two pEp users can be established, [I-D.marques-pep-rating], which describes the privacy indications that can be helpful for regular Internet users or [I-D.pep-keysync], which outlines pEp's peer-to-peer protocol to synchronize secret key material which belongs to the same account and user across various end-devices.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.3. Terms

The following terms are defined for the scope of this document:

- pEp Handshake: The process of one user contacting another over an independent channel in order to verify Trustwords (or fingerprints as a fallback). This can be done in-person or through established verbal communication channels, like a phone call. [I-D.marques-pep-handshake]

- Trustwords: A scalar-to-word representation of 16-bit numbers (0 to 65535) to natural language words. When doing a Handshake, peers are shown combined Trustwords of both public keys involved to ease the comparison. [I-D.birk-pep-trustwords]

- Trust On First Use (TOFU): cf. [RFC7435], which states: "In a protocol, TOFU calls for accepting and storing a public key or credential associated with an asserted identity, without authenticating that assertion. Subsequent communication that is authenticated using the cached key or credential is secure against an MiTM attack, if such an attack did not succeed during the vulnerable initial communication."

- Man-in-the-middle (MITM) attack: cf. [RFC4949], which states: "A form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data to masquerade as one or more of the entities involved in a communication association."

  Note: Historically, MITM has stood for '_Man_-in-the-middle'. However, to indicate that the entity in the middle is not always a human attacker, MITM can also stand for 'Machine-in-the-middle' or 'Meddler-in-the-middle'.

2. Opportunistic Security and Privacy for Email

In addition to the Protocol's Core Design Principles outlined in [I-D.birk-pep], the following sections on design principles are applicable to pEp for email applications.

2.1. Privacy by Default

The pEp formats and protocols aim to maximize privacy. Where privacy goals contradict with security goals, the privacy goals MUST take precedence.

Examples:

- pEp implementers MUST NOT make queries to public key servers by default. The reason for this is to make it more resource-intensive for centralized network actors to learn a user's social graph. This is also problematic security-wise, as centralized cryptographic key subversion at-scale is made easier. Instead, key distribution MUST be handled in-band while communicating with other peers.

- Trust information MUST NOT be attached to the communication partners' public keys. This is metadata which MUST be held locally and separately from the keys. Trust is established between the peers directly (peer-to-peer) and no trust information is held centrally (no support for the Web of Trust): that is, while pEp MUST be able to work with OpenPGP keys which carry trust information, this external trust information MUST NOT be used to signal any trust level to the pEp user.

- pEp-enabled MUAs MUST either engage in a signed-and-encrypted communication or in unsigned plaintext communication. While the signatures attached to plaintext messages can be verified, signed-only messages neither increase security nor privacy, so long as the corresponding public key is not authenticated.

2.2. Data Minimization

Data Minimization includes data sparsity and hiding of all technically concealable information whenever possible.

2.3. Metadata Protection

Email metadata (i.e., headers) MUST either be omitted or encrypted whenever possible.

The PGP/MIME specification as described in [RFC3156] provides few facilities for metadata protection: while the email body receives protection, the header section remains unprotected.

However, it is possible to also protect the information contained in the header field values by encapsulating the whole message into a MIME entity to be signed and encrypted.

The S/MIME Message Specification [RFC8551] defines a way to also protect the header section in addition to the content of a message:

The sending client MAY wrap a full MIME message in a message/rfc822 wrapper in order to apply S/MIME security services to header fields.

2.4. Interoperability

Implementers of pEp SHOULD be liberal in accepting non-pEp formats to encrypt email contents and metadata. pEp implementations MUST use the strict and interoperable pEp Email Format 1.0 (cf. Section 2.9.2) for any outgoing communication to non-pEp users. For communication between pEp users, more privacy-preserving formats (cf. Section 2.9) MUST be used. pEp Email Formats 2.0 and newer SHOULD NOT be used between users who are not recognized as pEp users (cf. Section 2.9.5), because for non-pEp users those formats are likely to produce unwanted visual artifacts.

2.5. End-to-End

For interpersonal messaging, an email endpoint in pEp is the MUA on a user's end-device: that is, encryption and decryption of messages MUST be executed on a user's end-device and MUST NOT depend on any third-party network infrastructure (i.e., any infrastructure outside a user's direct control).

[[*TODO*: Add enterprise settings with Key Escrow / Extra Keys]]

2.6. Peer-to-Peer

All relevant pEp mechanisms and state information about other peers MUST be held locally, on a peer's end-device. There MUST NOT be any reliance on an email server or even a centralized network component to hold relevant information for peers to be able to communicate or to authenticate themselves. Email servers (like, SMTP or IMAP) are only used as transport infrastructure for messages, but MUST not be relevant to hold actual state between peers.

[[TODO: Make clear there is a way that synchronizes trust in a peer-to-peer fashion, by using the Trust Sync mechanism.]]

2.7. User Experience (UX)

[[*TODO*: Add here what is specific to email]]

2.8. Identity System

In pEp for email, a user is a person or group who can have one or more identities, each represented by email addresses. Every identity has an own key attached to it. An email address can also be an alias

for an already existing identity, in which case the same key is attached to it.

All information about communication partners, like identities, keys and aliases MUST be held on a user's end-device as state information. This SHOULD be done using a structured format, to facilitate the synchronization of state information across various devices, taking into account multi-device scenarios, which are common today.

In pEp's reference implementation (cf. Section 11), keys are held using the key store of the cryptographic library, while peer-specific state information, including trust information, is held in a simple relational database.

[[*TODO*: Check optimal order for the following sections.]]

2.8.1. Address

In pEp for email, the SMTP address (e.g., mailto:alice@example.org) constitutes the network address.

2.8.2. Key

For now, a key in pEp for email is an OpenPGP key. Each identity has a default key attached for that identity. This is the public key to be used to encrypt communications to it.

2.8.3. User

A user in pEp for email is a specific person or group. A user has at least one identity, but can have more.

2.8.4. Identity

An identity in pEp for email is represented by an email URI, like mailto:alice@example.org. Identities are represented by email address URIs because a user may have multiple URIs. For example, if Alice uses mailto:alice@example.org for private purposes, but also wishes to have a public address, she may create another email address, such as anonymous@example.com. Because this is a new URI (mailto:anonymous@example.com), it is considered a new identity for Alice.

By default, pEp-enabled MUAs MUST generate a new key pair during new account configuration, so that a user's respective identities are not correlated to each other. However, if Alice wants her URIs to be handled as a single identity with one key, she may configure her respective identities as aliases.

For other email URIs pointing to the same identity, see the alias (cf. Section 2.8.5) concept.

2.8.5. Alias

Aliases share the same key and identity, e.g., the same key might be used for mailto:alice@example.org as well as for mailto:alice@example.net. That is, both addresses refer to the same identity.

2.9. pEp Email Formats

The pEp Email Formats 1.0, 2.0 and 2.1 are restricted MIME-based email formats, which ensure messages to be signed and encrypted. In accordance with pEp's privacy (and not security) focus, signed-only messages MUST NOT be produced (cf. Section 2.1). pEp-enabled clients MUST be able to render all pEp Email Formats properly: for outgoing communications, the most privacy-preserving format available is to be used, taking interoperability (cf. Section 2.4) into account.

Since pEp Email Format 2.0, a compatibility format (i.e., pEp Email Format 1.0, cf. Section 2.9.2) exists, which SHOULD be applied to non-pEp users, for which trustworthy public keys are available according to the local database.

In case no trustworthy encryption key is available, an unencrypted, unsigned MIME email is sent out. As in all pEp formats, also this (unprotected) message MUST contain the sender's public key, unless Passive Mode (cf. Section 7.1) is active.

All pEp Email Formats include a "pEpkey.asc" file attachment holding the sender's OpenPGP public key in ASCII-armored format, which is suitable for manual key import by non-pEp users. Thus, a user of any OpenPGP-enabled MUA is able to manually import the public key and engage in end-to-end encryption with the pEp sender. MUA implementers of PGP-capable email clients, even when not fully supporting pEp's protocols, are encouraged to automatically import the key such that the user can immediately engage in opportunistic encryption.

In pEp's reference implementation the subject is set to "pEp" (or alternatively to its UTF-8 representation as "=?utf-8?Q?p=E2=89=A1p?="). However, the subject's value of the outer message MUST be ignored. Therefore, the subject can be set to any value (e.g., "..." as used in other implementations).

2.9.1. Unencrypted pEp Format

This is the format to be used when unencrypted messages are sent out.

The unencrypted pEp format is a "multipart/mixed" MIME format, which by default ensures the delivery of the sender's public key as an attachment ("Content-Disposition: attachment").

A simple plaintext email looks like the following:

```
From: Alice <alice@example.org>
To: Bob <bob@example.org>
Date: Tue, 31 Dec 2019 05:05:05 +0200
X-pEp-Version: 2.1
MIME-Version: 1.0
Subject: Saying Hello
Content-Type: multipart/mixed; boundary="boundary"

--boundary
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: quoted-printable

Hello Bob

If you reply to this email using a pEp-enabled client, I will
be able to send you that sensitive material I talked to you
about.

Have a good day!

Alice

--
Sent with pEp for Android.

--boundary
Content-Type: application/pgp-keys; name="pEpkey.asc"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="pEpkey.asc"; size=2639

-----BEGIN PGP PUBLIC KEY BLOCK-----

[...]

-----END PGP PUBLIC KEY BLOCK-----

--boundary--
```

2.9.2. pEp Email Format 1.0

pEp Email Format 1.0 (PEF-1.0) is an encrypted and signed MIME format, which by default ensures:

- a signed and encrypted message, with subject encryption
- delivery of the sender's public key

PEF-1.0 has a "multipart/encrypted" MIME node on-the-wire format with an OpenPGP encrypted and signed filename "msg.asc", and "Content-Disposition: inline" attribute."

The subject is the only header field that can be protected with PEF-1.0. To achieve this protection, the real subject value is added to the top for the content section for the very first MIME entity with media type "text/plain", that is encrypted, e.g.:

```
Subject: Credentials
```

For example, an email with "Subject: Credentials" becomes "Subject: pEp" on-the-wire once encryption has taken place, with the true subject appended to the beginning of the message text once decrypted.

Thus, legacy clients not aware of nor able to parse pEp's subject encryption, still display the actual subject (in the above example: "Credentials") to the user. Whenever the first encrypted "text/plain" MIME entity contains such a subject line, the MUAs implementing pEp MUST render it to the user. Note that also the lines starting with "subject:" or "SUBJECT:" are to be rendered (as with header fields, this is case-insensitive).

A pEp-enabled MUA MUST add the "X-pEp-Version" header field with its highest value (preferably with value "2.1" as for pEp Email Format 2.1 Section 2.9.4) when producing this format. Here, a pEp-enabled MUA declares its capability to receive and render more privacy-preserving formats. Upgrading both sides to the highest version of the pEp Email Format allows pEp-enabled MUAs for best possible protection of metadata. For non-pEp MUAs it is OPTIONAL to add the "X-pEp-Version: 1.0" header field. However, this format is implicitly assumed even if this header field is not present.

Please note that for messages between pEp- and non-pEp clients the subject encryption MAY be disabled, sacrificing usability over privacy by avoiding artefacts for non-pEp recipients.

PEF-1.0 is also considered pEp's compatibility format towards non-pEp clients.

A PEF-1.0 example looks as follows:

```
From: Alice <alice@example.org>
To: Bob <bob@example.org>
Date: Wed, 1 Jan 2020 23:23:23 +0200
X-pEp-Version: 1.0
MIME-Version: 1.0
Subject: pEp
Content-Type: multipart/encrypted; boundary="boundary1";
 protocol="application/pgp-encrypted"

--boundary1
Content-Type: application/pgp-encrypted

Version: 1

--boundary1
Content-Type: application/octet-stream
Content-Transfer-Encoding: 7bit
Content-Disposition: inline; filename="msg.asc"

-----BEGIN PGP MESSAGE-----

[...]

-----END PGP MESSAGE-----

--boundary--
```

Decrypting the enclosed "msg.msc" part yields the following:

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="boundary2"
--boundary2
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: quoted-printable
Content-Disposition: inline; filename="msg.txt"

Subject: Credentials

Dear Bob

Please use "bob" with the following password to access the wiki site:

correcthorsebatterystaple

Please reach out if there are any issues and have a good day!

Alice

--boundary2
Content-Type: application/pgp-keys
Content-Disposition: attachment; filename="pEpkey.asc"

-----BEGIN PGP PUBLIC KEY BLOCK-----

[...]

-----END PGP PUBLIC KEY BLOCK-----

--boundary2--
```

Note that the user-intended subject value is encrypted in the first "text/plain" MIME entity under the "multipart/mixed" MIME node.

2.9.2.1. Deprecated variant of PEF-1.0

An earlier variant of PEF-1.0 started with a "multipart/mixed" MIME node, which in case of a simple text-only email without attachments and other MIME entities has

(1) a "text/plain" MIME entity with the PGP-encrypted content, and

(2) the sender's transferable public key at the very end.

This variant MUST NOT be produced anymore.

An example of this deprecated variant of PEF-1.0 looks as follows:

```
From: Alice <alice@example.org>
To: Bob <bob@example.org>
Date: Wed, 1 Jan 2020 23:23:23 +0200
X-pEp-Version: 1.0
MIME-Version: 1.0
Subject: pEp
Content-Type: multipart/mixed; boundary="boundary"

--boundary
Content-Type: application/pgp-encrypted

Version: 1

--boundary
Content-Type: text/plain; charset="utf8"
Content-Transfer-Encoding: 7bit

-----BEGIN PGP MESSAGE-----
[...]
-----END PGP MESSAGE-----

--boundary
Content-Type: application/pgp-keys; name="pEpkey.asc"
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="pEpkey.asc"

-----BEGIN PGP PUBLIC KEY BLOCK-----

[...]

-----END PGP PUBLIC KEY BLOCK-----

--boundary--
```

There, decrypting the PGP encrypted text/plain element yields a text like the following; most obviously, the intended subject line is now visible:

Subject: Credentials

Dear Bob

Please use "bob" with the following password to access the wiki site:

correcthorsebatterystaple

Please reach out if there are any issues and have a good day!

Alice

2.9.3. pEp Email Format 2.0

pEp Email Format 2.0 (PEF-2.0) is a strict MIME format, which by default ensures:

- a signed and encrypted message, with full email encapsulation
- delivery of the sender's public key

In PEF-2.0, the actual email (inner message) is encapsulated by a MIME entity ("Content-Type: message/rfc822"), which is the second part of a "multipart/mixed" MIME node. The first part of this MIME node contains a "text/plain" MIME entity, including a marker text "pEp-Message-Wrapped-Info: OUTER" (in its MIME content). This is used for proper displaying and mapping of the nested message and its encrypted header fields. Like with the PEF-1.0 (cf. Section 2.9.2), the third (and last) part of the "multipart/mixed" MIME node MUST contain the sender's public key.

The "multipart/mixed" MIME node is encrypted inside yet another MIME node ("Content-Type: multipart/encrypted", cf. [RFC1847] / [RFC3156]), which is the body part of the outer message.

Thus, the whole header section of the inner message can be fully preserved, not only encrypted, but also signed. In the outer message, however, when communicating with pEp users all header fields that are not needed MUST be omitted to the fullest extent possible.

Once encrypted, only the outer message consisting of the (minimal) outer header section and the "multipart/encrypted" MIME entity as body with an application/octet-stream "Content-Type" with name "msg.asc" is visible on the wire.

If the receiving side is not a known pEp-enabled MUA, but there is a trustworthy public key available, PEF-1.0 (cf. Section 2.9.2) MUST be used to send the email.

In any case, the "X-pEp-Version" header field MUST be set to version 2.0, as the highest version that the sender supports.

The following example shows a PEF-2.0 multipart/encrypted email, signed and encrypted, as an 7bit octet stream with a filename "msg.asc", with "Content-Disposition: inline". Within that, the original email message is fully contained in encrypted form (like this, also the subject line gets encrypted). The support of version 2.0 is announced in the "X-pEp-Version" header field (in this example, 2.0 is the newest pEp Email Format the pEp-enabled MUA is able to produce and render):

```
From: Alice <alice@example.org>
To: Bob <bob@example.org>
Date: Wed, 1 Jan 2020 23:23:23 +0200
X-pEp-Version: 2.0
MIME-Version: 1.0
Subject: pEp
Content-Type: multipart/encrypted; boundary="boundary1";
 protocol="application/pgp-encrypted"

--boundary1
Content-Type: application/pgp-encrypted

Version: 1

--boundary1
Content-Type: application/octet-stream
Content-Transfer-Encoding: 7bit
Content-Disposition: inline; filename="msg.asc"

-----BEGIN PGP MESSAGE-----

[...]

-----END PGP MESSAGE-----

--boundary--
```

Decrypting "msg.asc" results in a multipart/mixed node, with three elements:

(1) a text part indicating this is the encapsulated message

(2) the original message encapsulated by a "message/rfc822" MIME entity, and

(3) the transferable sender's public key in ASCII-armored format.

An unwrapped example looks like this:

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="boundary2"

--boundary2
Content-Type: text/plain; charset="utf-8"
Content-Disposition: inline; filename="msg.txt"

pEp-Wrapped-Message-Info: OUTER

--boundary2
Content-Type: message/rfc822

Message-ID: <pEp.1234>
Date: Wed, 1 Jan 2020 23:23:23 +0200
Subject: Credentials
X-pEp-Version: 2.0
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="boundary3"

--boundary3
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: quoted-printable

pEp-Wrapped-Message-Info: INNER

Dear Bob

Please use "bob" with the following password to access the wiki site:

correcthorsebatterystaple

Please reach out if there are any issues and have a good day!

Alice

--boundary3--

--boundary2

Content-Type: application/pgp-keys
Content-Disposition: attachment; filename="pEpkey.asc"

-----BEGIN PGP PUBLIC KEY BLOCK-----

[...]
```

```
-----END PGP PUBLIC KEY BLOCK-----

--boundary2--
```

2.9.4. pEp Email Format 2.1

pEp Email Format 2.1 (PEF-2.1) introduces further pEp-specific header fields to the inner message, which help to determine the behavior between pEp users.

In normal interpersonal messaging those additional header fields are:

(1) "X-pEp-Wrapped-Message-Info: INNER" header field stating that the message carrying this is to be considered the most inner message containing the original email (this is particularly relevant for mixnet or other scenarios of nested messaging; cf. [pEp.mixnet])

(2) "X-pEp-Sender-FPR" header field with the value set to sender's full 160-bit public key fingerprint (e.g., "1234567890ABCDEF1234567890ABCDEF12345678"), and

(3) the "X-pEp-Version" header field set to version "2.1".

As with PEF-2.0 Section 2.9.3, in PEF-2.1 the actual email (inner message) is encapsulated by a MIME entity ("Content-Type: message/rfc822"), which is the second part of a "multipart/mixed" MIME node. The first part of this MIME node contains a "text/plain" MIME entity, which SHOULD be used to inform about the nature of this format (in case a non-pEp client encounters in the mailbox). It MAY be used to carry the intended subject of the inner message (which is not done in current reference implementations). Like with the PEF-1.0 (cf. Section 2.9.2) and PEF 2.0 (cf. Section 2.9.3), the third (and last) part of this "multipart/mixed" MIME node MUST contain the sender's public key.

This "multipart/mixed" MIME node is encrypted inside yet another MIME node ("Content-Type: multipart/encrypted", cf. [RFC1847] / [RFC3156]), which is the body part of the outer message.

A caveat of PEF-2.1 is that message rendering varies considerably across different MUAs. This is relevant as it might happen that a non-pEp MUA encounters a PEF-2.1 message (e.g., if a pEp-enabled client was used in the past). No standard is currently available which enables MUAs to reliably determine whenever a nested "message/rfc822" MIME entity is meant to render the contained email message, or if it was effectively intended to be forwarded as an attachment, where a user needs to click on in order to see its content. To help unaware MUAs, a Content-Type header field parameter with name

"forwarded" as per [I-D.melnikov-iana-reg-forwarded] is added to the Content-Type header field. MUAs can use this to distinguish between a forwarded message and a nested message (i.e., using "forwarded=no").

When the receiving peer was registered as being only PEF-2.0-capable, the message must be sent in PEF-2.0 (cf. Section 2.9.3). The reason for this is that pEp-enabled MUAs which are only PEF-2.0-capable rely on the plaintext "pEp-Message-Wrapped-Info: OUTER" and "pEp-Message-Wrapped-Info: INNER" markers to properly display and map the nested message and its encrypted header fields.

As with PEF-1.0, if the receiving side is not a known pEp-enabled MUA, but there is a trustworthy public key available, PEF-1.0 (cf. Section 2.9.2) MUST be used to send the email.

In any case, the "X-pEp-Version" header field MUST be set to version 2.1, as the highest version that the sender supports.

This is an example of what the format looks like between two PEF-2.1-capable clients:

```
From: Alice <alice@example.org>
To: Bob <bob@example.org>
Date: Wed, 1 Jan 2020 23:23:23 +0200
X-pEp-Version: 2.1
MIME-Version: 1.0
Subject: pEp
Content-Type: multipart/encrypted; boundary="boundary1";
              protocol="application/pgp-encrypted"

--boundary1
Content-Type: application/pgp-encrypted

Version: 1

--boundary1
Content-Type: application/octet-stream
Content-Transfer-Encoding: 7bit
Content-Disposition: inline; filename="msg.asc"

-----BEGIN PGP MESSAGE-----

[...]

-----END PGP MESSAGE-----

--boundary1--
```

Unwrapping the "multipart/encrypted" MIME node, yields this:

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="boundary2"

--boundary2
Content-Type: text/plain; charset="utf-8"
Content-Disposition: inline; filename="msg.txt"

This message was encrypted with pEp (https://pep.software). If you
are seeing this message, your client does not support raising message
attachments. Please click on the message attachment to view it,
or better yet, consider using pEp!

--boundary2
Content-Type: message/rfc822; forwarded="no"

Message-ID: <pEp.1234>
Date: Wed, 1 Jan 2020 23:23:23 +0200
Subject: Credentials
X-pEp-Version: 2.1
X-pEp-Wrapped-Message-Info: INNER
X-pEp-Sender-FPR: 1234567890ABCDEF1234567890ABCDEF12345678
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="boundary3"

--boundary3
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: quoted-printable

Dear Bob

Please use "bob" with the following password to access the wiki site:

correcthorsebatterystaple

Please reach out if there are any issues and have a good day!

Alice

--boundary3--

--boundary2

Content-Type: application/pgp-keys
Content-Disposition: attachment; filename="pEpkey.asc"

-----BEGIN PGP PUBLIC KEY BLOCK-----
```

```
[...]

-----END PGP PUBLIC KEY BLOCK-----

--boundary2--
```

[[TODO: Clarify on the "raising message" term in the example.]]

2.9.5. Protocol Negotiation for Format Selection

To be able to decide which email format to generate, the pEp-enabled MUA REQUIRES to record state on a per-identity basis. Once a "X-pEp-Version" header field is discovered, the user MUST be recorded as a pEp user and the corresponding pEp Version it supports (according to the highest value of the "X-pEp-Version" header field encountered).

2.9.6. Saving Messages

In accordance with the Privacy by Default principle, messages sent or received in encrypted form MUST be saved with the identity's respective public key.

Messages sent or received in unencrypted form, SHOULD NOT be saved in encrypted form on the email server: this reflects the Privacy Status the user encountered when sending or receiving the email and thus meets the user's expectations.

Instead, message drafts MUST always be saved with the identity's public key.

Other messages sent and received MUST be saved encrypted by default: for most end-user scenarios, the servers users work with, are considered untrusted.

For trusted environments (e.g., in organizations) and to conform to legally binding archiving regulations, pEp implementations MUST provide a "Trusted Server" option. With the user's explicit consent (opt-in), unencrypted copies of the Messages MUST be held on the mail servers controlled by the organization.

3. Key Management

3.1. Key Generation

A pEp-enabled Mail User Agent MUST consider every email account as an new identity: for each identity, a different key pair MUST be created automatically if no key material with sufficient length is available. By default, RSA-4096 key pairs for OpenPGP encryption [RFC4880]

SHOULD be generated automatically for each email account. However, the key length MUST be at least 2048 bits. Elliptic curve keys with at least 256 bits MUST be supported, but SHOULD NOT yet be generated and announced by default for interoperability reasons.

If for an identity there's an RSA key pair with less than 2048 bits, new keys MUST be generated.

3.2. Private Keys

[[TODO: Add here what is specific to email]]

3.2.1. Storage

[[TODO: Add here what is specific to email]]

3.2.2. Passphrase

[[TODO: Add here what is specific to email]]

3.2.3. Private Key Export / Import

3.3. Public Key Distribution

By default, public keys MUST always be attached to any outgoing message as described in Section 2.9. If this is undesired, Passive Mode (cf. Section 7.1) can be activated.

3.4. Key Reset

[[TODO: Add here what is specific to email]]

4. Trust Management

The following example roughly describes a pEp email scenario with a typical initial message flow to demonstrate key exchange and basic trust management:

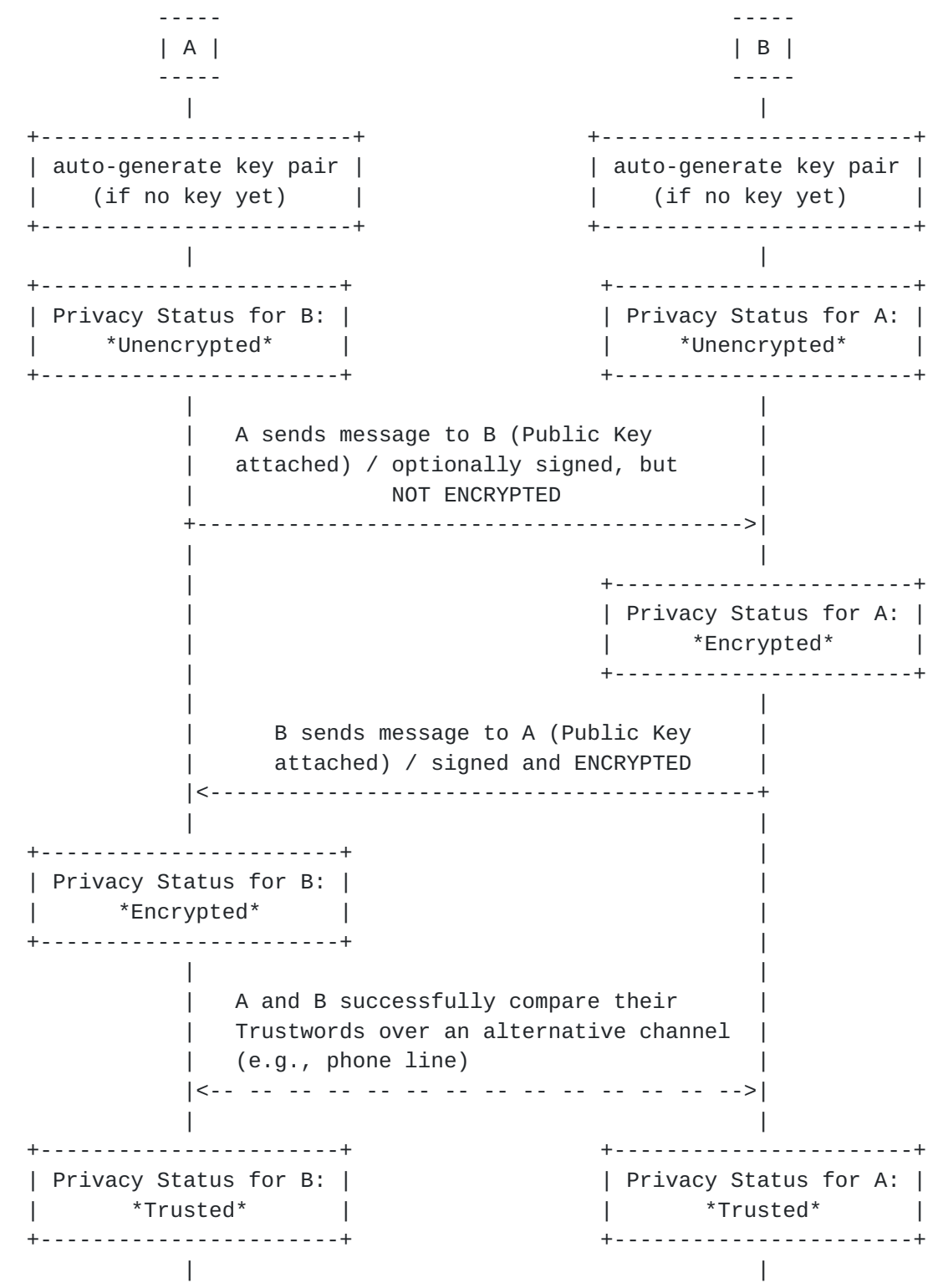
1. Alice - knowing nothing of Bob - sends an email to Bob. As Alice has no public key from Bob, this email is sent out unencrypted. However, Alice's public key is automatically attached.

2. Bob can just reply to Alice and - as he received her public key - his MUA is now able to encrypt the message. At this point, the rating for Alice changes to "encrypted" in Bob's MUA, which (UX-wise) can be displayed using yellow color (cf. Section 4.3).

3. Alice receives Bob's key. As of now Alice is also able to send secure emails to Bob. The rating for Bob changes to "encrypted" (with yellow color) in Alice's MUA (cf. Section 4.3).

4. If Alice and Bob want to prevent man-in-the-middle (MITM) attacks, they can engage in a pEp Handshake comparing their so-called Trustwords (cf. Section 4.2) and confirm this process if those match. After doing so, their identity rating changes to "encrypted and authenticated" (cf. Section 4.3), which (UX-wise) can be displayed using a green color.

As color code changes for an identity, this is also reflected to future messages to/from this identity. Past messages, however, MUST NOT be altered.



[[TODO: Add more of what is specific to email]]

4.1. Privacy Status

[[TODO: Add here what is specific to email]]

4.2. Handshake

[[TODO: Add here what is specific to email]]

4.3. Trust Rating

[[TODO: Add here what is specific to email]]

5. Synchronization

As per [I-D.pep-keysync]:

[[TODO: Add here what is specific to email]]

5.1. Private Key Synchronization

[[TODO: Add here what is specific to email]]

5.2. Trust Synchronization

[[TODO: Add here what is specific to email]]

6. Interoperability

[[TODO: Add here what is specific to email]]

7. Options in pEp

7.1. Option "Passive Mode"

In email, Passive Mode primarily exists as an option to avoid potential usability issues in certain environments where Internet users might get confused by the exposure of public keys in email attachments. In principle, however, this "problem" can be mitigated either by training or by MUA implementers displaying public key material in a more symbolic way or even importing it automatically and then hiding this attachment altogether (as pEp implementers are supposed to do, such that regular Internet users do not have to bother about keys).

Passive Mode has a negative impact on privacy: additional unencrypted message exchanges are needed until pEp's by-default encryption can take place.

Passive Mode MUST only affect unencrypted communications and MUST be inactive by default. By opting in to Passive Mode, the sender's public key MUST NOT be attached when sending out unsecure emails. On the other hand, Passive Mode is without any effect when pEp is able to send out an encrypted message, because the necessary encryption key(s) are available.

In this situation, opportunistic by-default encryption MUST take place: there, the sender's public key is attached in encrypted form as constituent part of one of pEp's PGP/MIME-based message format described in Section 2.9.

Additionally, Passive Mode MUST be without effect, if a receiver learns that an MUA is actually pEp-capable, even if the sender involved is in Passive Mode, too: this MUST be recognized by the "X-pEp-Version" header field, as the only clear indicator to detect pEp users. That means that a pEp-enabled MUA is REQUIRED to attach its corresponding public key to another pEp user in any case, such that they can engage in opportunistic encryption.

[[TODO: Add message examples and a flow chart, if needed]]

7.2. Option "Disable Protection"

This is an opt-in mechanism to enforce that messages go out unprotected. Even if encryption keys for recipient(s) are available, this option MUST enforce that messages are sent in the Section 2.9.1 format.

7.3. Option "Extra Keys"

[[TODO: Add here what is specific to email]]

7.4. Option "Blacklist Keys"

[[TODO: Add here what is specific to email]]

7.5. Option "Trusted Server"

[[TODO: Add here what is specific to email]]

8. Security Considerations

[[TODO]]

9. Privacy Considerations

[[TODO]]

10. IANA Considerations

This document has no actions for IANA.

11. Implementation Status

11.1. Introduction

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "[...] this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit."

11.2. Current software implementations of pEp

The following software implementations of the pEp protocols (to varying degrees) already exists:

- pEp for Outlook as add-on for Microsoft Outlook, release [SRC.pepforoutlook]
- pEp for iOS (implemented in a new MUA), release [SRC.pepforios]
- pEp for Android (based on a fork of the K9 MUA), release [SRC.pepforandroid]
- pEp for Thunderbird as a new add-on for Thunderbird, beta [SRC.pepforthunderbird]

- Enigmail/pEp as add-on for Mozilla Thunderbird, release (will be discontinued late 2020/early 2021) [SRC.enigmailpep]

pEp for Android, iOS, Outlook and Thunderbird are provided by pEp Security, a commercial entity specializing in end-user pEp implementations while Enigmail/pEp is pursued as community project, supported by the pEp Foundation.

All software is available as Free and Open Source Software and published also in source form.

12. Acknowledgements

Special thanks go to Krista Bennett and Volker Birk for the reference implementation on pEp and the ideas leading to this draft.

The author would like to thank the following people who provided substantial contributions, helpful comments or suggestions for this document: Berna Alp, Kelly Bristol, Bernie Hoeneisen, Claudio Luck and Lars Rohwedder.

This work was initially created by the pEp Foundation, and was initially reviewed and extended with funding by the Internet Society's Beyond the Net Programme on standardizing pEp. [ISOC.bnet]

13. References

13.1. Normative References

[I-D.birk-pep]
Birk, V., Marques, H., and B. Hoeneisen, "pretty Easy privacy (pEp): Privacy by Default", draft-birk-pep-05 (work in progress), November 2019.

[I-D.marques-pep-handshake]
Marques, H. and B. Hoeneisen, "pretty Easy privacy (pEp): Contact and Channel Authentication through Handshake", draft-marques-pep-handshake-05 (work in progress), July 2020.

[I-D.marques-pep-rating]
Marques, H. and B. Hoeneisen, "pretty Easy privacy (pEp): Mapping of Privacy Rating", draft-marques-pep-rating-03 (work in progress), January 2020.

[I-D.melnikov-iana-reg-forwarded]
Melnikov, A. and B. Hoeneisen, "IANA Registration of Content-Type Header Field Parameter 'forwarded'", draft-melnikov-iana-reg-forwarded-00 (work in progress), November 2019.

[RFC1847] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, DOI 10.17487/RFC1847, October 1995, <https://www.rfc-editor.org/info/rfc1847>.

[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>.

[RFC3156] Elkins, M., Del Torto, D., Levien, R., and T. Roessler, "MIME Security with OpenPGP", RFC 3156, DOI 10.17487/RFC3156, August 2001, <https://www.rfc-editor.org/info/rfc3156>.

[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>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <https://www.rfc-editor.org/info/rfc7435>.

13.2. Informative References

[I-D.birk-pep-trustwords]
Hoeneisen, B. and H. Marques, "IANA Registration of Trustword Lists: Guide, Template and IANA Considerations", draft-birk-pep-trustwords-05 (work in progress), January 2020.

[I-D.pep-keysync]
Birk, V., Hoeneisen, B., and K. Bristol, "pretty Easy privacy (pEp): Key Synchronization Protocol (KeySync)", draft-pep-keysync-02 (work in progress), July 2020.

[ISOC.bnet]
Simao, I., "Beyond the Net. 12 Innovative Projects Selected for Beyond the Net Funding. Implementing Privacy via Mass Encryption: Standardizing pretty Easy privacy's protocols", June 2017, <https://www.internetsociety.org/blog/2017/06/12-innovative-projects-selected-for-beyond-the-net-funding/>.

[pEp.mixnet]
pEp Foundation, "Mixnet", June 2020, <https://dev.pep.foundation/Mixnet>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

[SRC.enigmailpep]
"Source code for Enigmail/pEp", November 2020, <https://enigmail.net/index.php/en/download/source-code>.

[SRC.pepforandroid]
"Source code for pEp for Android", November 2020, <https://pep-security.lu/gitlab/android/pep>.

[SRC.pepforios]
"Source code for pEp for iOS", November 2020, <https://pep-security.ch/dev/repos/pEp_for_iOS/>.

[SRC.pepforoutlook]
"Source code for pEp for Outlook", November 2020, <https://pep-security.lu/dev/repos/pEp_for_Outlook/>.

[SRC.pepforthunderbird]
"Source code for pEp for Thunderbird", November 2020, <https://pep-security.lu/dev/repos/pEp_for_Thunderbird/>.

Appendix A. Document Changelog

[[RFC Editor: This section is to be removed before publication]]

- draft-pep-email-01:
  - Minor language improvements
- draft-pep-email-00:
  - Major restructure of the document
  - Major fixes in the description of the various message formats
  - Add many open questions and comments inline (TODO)
  - Add IANA Considerations section
  - Change authors and acknowledgment section
  - Add internal references
  - Describe Passive Mode
  - Better explanation on how this document relates to other pEp documents
- draft-marques-pep-email-02:
  - Add illustrations
  - Minor fixes
  - Add longer list of Open Issues (mainly by Bernie Hoeneisen)
- draft-marques-pep-email-01:
  - Remove an artefact, fix typos and minor editorial changes; no changes in content
- draft-marques-pep-email-00:
  - Initial version

Appendix B. Open Issues

[[RFC Editor: This section should be empty and is to be removed before publication]]

- Eventually move the many TODOs into this Open Issues section.
- Describe KeyImport to induce the import from secret keys from other devices
- Describe / Reference KeySync (and other sync, through IMAP)
- Add key pair revocation strategy
- Create clearer relations to the pEp rating draft (draft-marques-pep-rating), as this plays an important role in how Messages are rendered and how they need to be presented (after rating) for a user to have awareness about his privacy status in any given situation.
- Make document more coherent: check with pEp's general draft pieces to fill on both sides and how to reference them vice-versa (this is now pending on the reworked general draft to be published).
- Elaborate more on the X-EncStatus header field and for Trusted Server situations / mirrors and describe operations.
- Explain Key Mapping (between OpenPGP and S/MIME)

Author's Address

Hernani Marques
pEp Foundation
Oberer Graben 4
CH-8400 Winterthur
Switzerland

Email: hernani.marques@pep.foundation
URI: https://pep.foundation/