Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 19 June 2023

V. Birk
H. Marques
B. Hoeneisen
pEp Foundation

16 December 2022

pretty Easy privacy (pEp): Privacy by Default
draft-pep-general-02

Abstract

The pretty Easy privacy (pEp) model and protocols describe a set of conventions for the automation of operations traditionally seen as barriers to the use and deployment of secure, privacy-preserving end-to-end messaging. These include, but are not limited to, key management, key discovery, and private key handling (including peer-to-peer synchronization of private keys and other user data across devices). Human Rights-enabling principles like data minimization, end-to-end and interoperability are explicit design goals. For the goal of usable privacy, pEp introduces means to verify communication between peers and proposes a trust-rating system to denote secure types of communications and signal the privacy level available on a per-user and per-message level. Significantly, the pEp protocols build on already available security formats and message transports (e.g., PGP/MIME with email), and are written with the intent to be interoperable with already widely-deployed systems in order to ease adoption and implementation. This document outlines the general design choices and principles of pEp.

Table of Contents

1. Introduction

Secure and private communications are vital for many different reasons, and there are particular properties that privacy-preserving protocols need to fulfill in order to best serve users. In particular, [RFC8280] has identified and documented important principles such as data minimization, the end-to-end principle, and interoperability as integral properties which enable access to Human Rights. Today's applications widely lack privacy support that ordinary users can easily adapt. The pretty Easy privacy (pEp) protocols generally conform to the principles outlined in [RFC8280],

and, as such, can facilitate the adoption and correct usage of secure and private communications technology.

The pretty Easy privacy (pEp) protocols are propositions to the Internet community to create software for peers to automatically encrypt, anonymize (where possible), and verify their daily written digital communications. This is achieved by building upon already existing standards and tools to automate each step a user needs to carry out in order to engage in secure end-to-end encrypted communications. Significantly, the pEp protocols describe how to achieve this without dependence on centralized infrastructures.

The pEp project emerged from the CryptoParty movement. During that time, the initiators learned that while step-by-step guides can help some users to engage in secure end-to-end communications; it is both more effective and convenient for the vast majority of users if these step-by-step guides are put into running code (following a protocol), which automates the initial configuration and general usage of cryptographic tools. To facilitate this goal, pEp proposes the automation of key management, key discovery, and key synchronization through an in-band approach that follows the end-to-end principle.

To mitigate man-in-the-middle (MITM) attacks by an active adversary, and as the only manual step users carry out in the course of the protocols, pEp proposes a Trustword [I-D.pep-trustwords] mechanism. Trustwords are natural language representations of two peers' fingerprints. Users can verify their trust in a paired communication channel by comparing the corresponding Trustwords.

The privacy-by-default principles that pEp introduces are in accordance with the perspective outlined in [RFC7435], aiming to provide Opportunistic Security in the sense of "some protection most of the time". This is done, however, with the subtle but important difference that when privacy is weighed against security, the choice defaults to privacy. Therefore, data minimization is a primary goal in pEp (e.g., hiding subject lines and headers unnecessary for email transport inside the encrypted payload of a message).

The pEp propositions are focused on (but not limited to) written digital communications and cover asynchronous (offline) types of communications like email as well as synchronous (online) types such as chat.

pEp's goal is to bridge different standardized and widely-used communication channels such that users can reach communication partners in the most privacy-enhancing way possible.

1.1. Relationship to Other pEp Documents

While this document outlines the general design choices and principles of pEp, other related documents specialize in more particular aspects of the model, or the application of pEp on a specific protocol like as follows:

1. pEp-enabled applications (e.g., pEp email [I-D.pep-email]).

2. Helper functions for peer interaction, which facilitate understanding and handling of the cryptographic aspects of pEp implementation for users (e.g., pEp Handshake [I-D.pep-handshake]).

3. Helper functions for interactions between a user's own devices, which give the user the ability to run pEp applications on different devices at the same time, such as a computer, mobile phone, or tablets (e.g., pEp KeySync [I-D.pep-keysync]).

In addition, there are documents that do not directly depend on this one, but provide generic functions needed in pEp, e.g., IANA Registration of Trustword Lists [I-D.pep-trustwords].

[[Note: At this stage it is not yet clear to us how many of our implementation details should be part of new RFCs and where we can safely refer to already existing RFCs to clarify which RFCs we rely on.]]

1.2. 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.

1.3. 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.pep-handshake]

* Trustwords: A representation of 16-bit natural numbers (0 to 65535) as natural language words: For each natural language a fixed number-to-word map can be defined as convention and registered with IANA. Trustwords are generated from the combined public key fingerprints of a both communication partners. Trustwords are used for verification and establishment of trust (for the respective keys and communication partners). [I-D.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. Protocol's Core Design Principles

2.1. Privacy by Default

pEp's most important goal is to ensure privacy above all else. To clarify, pEp's protocol defaults are designed to maximize both security and privacy, but in the few cases where achieving both more privacy and more security are in conflict, pEp chooses more privacy.

In contrast to pEp's prioritization of user privacy, OpenPGP's Web-of-Trust (WoT) releases user and trust level relationships to the public. In addition, queries to OpenPGP keyservers dynamically disclose the social graph, indicating a user's intent to communicate with specific peers. Similar issues exist in other security protocols that rely upon a centralized trust model, such as PKIX [RFC5280] used e.g., for S/MIME [RFC8551].

2.2. Data Minimization

Data minimization keeps data spare and hides all technically concealable information whenever possible. It is an important design goal of pEp.

2.3. Interoperability

The proposed pEp protocols seek interoperability with established message formats, as well as cryptographic security protocols and their widespread implementations.

To achieve this interoperability, pEp follows Postel's Robustness Principle outlined in [RFC1122]: "Be liberal in what you accept, and conservative in what you send."

Particularly, pEp applies Postel's principle as follows:

* pEp is conservative (strict) in requirements for pEp implementations and how they interact with pEp or other compatible implementations.

* pEp liberally accepts input from non-pEp implementations. For example, in email, pEp will not produce outgoing messages, but will transparently support decryption of incoming PGP/INLINE messages.

* Finally, where pEp requires divergence from established RFCs due to privacy concerns (e.g., from OpenPGP propositions as defined in [RFC4880], options SHOULD be implemented which empower the user to override pEp's defaults.

2.4. End-to-End

Because of the inherent privacy risks in using remote or centralized infrastructures, implementations of pEp messaging, by default, MUST NOT obtain content and information from remote or centralized locations, as this constitutes a privacy breach. In email this issue exists with HTML mails.

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).

2.5. Peer-to-Peer

Messaging and verification processes in pEp are designed to work in a peer-to-peer (P2P) manner, without the involvement of intermediaries.

This means there MUST NOT be any pEp-specific central services whatsoever needed for pEp implementations, both in the case of verification of peers and for the actual encryption.

However, implementers of pEp MAY provide options for interoperation with providers of centralized infrastructures (e.g., to enable users to communicate with their peers on platforms with vendor lock-in).

Trust provided by global Certificate Authorities (e.g., commercial X.509 CAs) MUST NOT be signaled as trustworthy (cf. [I-D.pep-rating]) to users of pEp (e.g., when interoperating with peers using S/MIME) by default.

2.6. User Interaction

Implementers of pEp SHOULD NOT expose users to technical terms and views, especially those specific to cryptography, like "keys", "certificates", or "fingerprints". However, certain (advanced) users MAY explicitly opt-in to see such terms, e.g., seeking for more options. Advanced settings may be available. In some cases, such settings or options may be required.

The authors believe that widespread adoption of end-to-end cryptography is possible if users are not required to understand cryptography and key management. This belief forms the central goal of pEp, which is that users can simply rely on the principles of Privacy by Default.

On the other hand, to preserve usability, users SHOULD NOT be required to wait for cryptographic tasks such as key generation to complete before being able to use their respective message client for its default purpose. In short, pEp implementers MUST ensure that the ability to draft, send, and receive messages is always preserved, even if that means a message is sent unencrypted, in accordance with the Opportunistic Security approach outlined in [RFC7435].

In turn, pEp implementers MUST ensure that a distinguishable privacy status is clearly visible to the user, both on a per-contact as well as per-message level. This allows users to assess both the privacy level for the message and the trust level of its intended recipients before choosing to send it.

[[*NOTE*: We are aware of the fact that usually UX requirements are not part of RFCs. However, in order to encourage massive adoption of secure end-to-end encryption while at the same time avoiding putting users at risk, we believe certain straightforward signaling requirements for users to be a good idea, just as it is currently done for already-popular instant messaging services.]]

3. pEp Identity System

Everyone has the right to choose how to reveal themselves to the world, both offline and online. This is an important element to maintain psychological, physical, and digital privacy. As such, pEp users MUST have the option to choose their identity, and they MUST have the ability to maintain multiple identities.

These different identities MUST NOT be externally correlatable with each other by default. On the other hand, combining different identities when such information is known MUST be supported (alias support).

3.1. User

A pEp User is a real world human being (or a group of human beings). If it is a single human being, it can be called person.

A pEp User ID (user_id) identifies a pEp User. The variable user_id SHOULD be a Universally Unique IDentifier (UUID), however it MAY also be an arbitrary unique string.

A pEp User may have a default pEp Key (cf. Section 3.3).

In the pEp reference implementation (cf. Appendix A), the 'id' (text) field in table 'person' contains the pEp User ID. Table 'identity' (that references to table 'person') the 'user_id' (text) field contains the pEp User ID, too. The 'main_key_id' (text) field in table 'person' contains the default pEp Key (that references to field 'fpr' in table 'pgp_keypair'; cf. Section 3.3).

3.1.1. Own User

The Own User in pEp is a special case of a pEp User (cf. Section 3.1), i.e. the (local) pEp User that utilizes the pEp client in question. As all pEp Users also the Own User needs a User ID, which is assigned at creation of the Own User entry.

In the pEp reference implementation (cf. Appendix A), the 'is_own' (integer) field in table 'identity' is set to 1, if the entry belongs to an Own User.

3.2. Address

An address in pEp means the designator of a destination where messages can be routed to and accessed from, e.g., email address, Uniform Resource Identifier (URI), Network Access Identifier (NAI), phone number, etc.

An address may belong to one or more users. A user may have multiple Addresses.

[[Note: It might be necessary to introduce further addressing schemes through IETF contributions or IANA registrations, e.g., implementing pEp to bridge to popular messaging services with no URIs defined.]]

In the pEp reference implementation (cf. Appendix A), the 'address' (text) field in table 'identity' contains the address.

3.3. Key

A pEp Key is an asymmetric cryptographic key pair compatible to OpenPGP [RFC4880].

pEp Keys are identified by the full fingerprint (fpr) of the public key. The fingerprint is obtained from the specific cryptographic application used to handle the keys. The canonical representation of the fingerprint in pEp is upper-case hexadecimal with zero-padding and no separators or spaces.

In the pEp reference implementation (cf. Appendix A), the 'fpr' (text) field in table 'pgp_keypair' contains the fingerprint of the pEp Key. The 'main_key_id' (text) field in tables 'person' and 'identity' (containing the default pEp Key) links to field 'fpr' in table 'pgp_keypair').

3.4. Identity

A pEp Identity is a representation of a pEp User (cf. Section 3.1), defining how this pEp User appears within the network of a messaging system. This representation may or may not be pseudonymous in nature.

A pEp Identity is defined by mapping a pEp User ID (cf. Section 3.1) to an address (cf. Section 3.2).

A pEp Identity can have a temporary pEp User ID (user_id) as a placeholder until a real pEp User ID is known.

In case different accounts are utilized by the same pEp User, a different pEp Key (cf. Section 3.3) for each account MUST be created. This allows a pEp User to retain different identities that cannot be correlated by sharing the same key for those identities (improved privacy).

A pEp User as well as each pEp Identity MAY have a default pEp Key. When both - a pEp Identity and the related pEp User - have a default pEp Key assigned, the pEp Identity's default pEp Key MUST take precedence over the pEp User's default pEp Key.

In the pEp reference implementation (cf. Appendix A), the above described mapping is done by adding an entry to table 'identity' linking its field 'user_id' to the 'id' field in table 'person'.

3.5. Alias

pEp Aliases share the same key and identity.

4. Key Management

In order to achieve the goal of widespread adoption of secure communications, key management in pEp MUST be automated.

4.1. Key Generation

A pEp implementation MUST ensure that cryptographic keys (cf. Section 3.3) for every configured pEp Identity (cf. Section 3.4) are available. If a corresponding key pair for the identity of a pEp User is found and said pEp Identity fulfills the requirements (e.g., for email, as set out in [I-D.pep-email]), said key pair SHOULD be reused. Otherwise a new key pair MUST be generated. This may be carried out instantly upon its configuration.

On devices with limited processing power (e.g., mobile devices) the key generation may take more time than a user is willing to wait. If this is the case, users SHOULD NOT be stopped from communicating, i.e., the key generation process SHOULD be carried out in the background.

4.2. Private Keys

4.2.1. Storage

Private keys in pEp implementations MUST always be held on the end user's device(s): pEp implementers MUST NOT rely on private keys stored in centralized remote locations. This applies even for key storages where the private keys are protected with sufficiently long passphrases. It is considered a violation of pEp's P2P design principle to rely on centralized infrastructures (cf. Section 2.5). This also applies for pEp implementations created for applications not residing on a user's device (e.g., web-based MUAs). In such cases, pEp implementations MUST be designed in a way such that the locally-held private key can neither be directly accessed by the server application nor transfered to any server, i.e. the private key MUST remain local and MUST NOT be leaked anywhere else.

Furthermore, it is important that browser add-ons implementing pEp functionality do not dynamically obtain their cryptographic code from a centralized (e.g., cloud) service. Cryptographic code MUST always be stored locally, installed e.g., as part of a certified and signed installation package, containing the add-on and additional code to execute the cryptographic functions. Dynamically obtained code is considered a centralized attack vector allowing for backdoors, negatively impacting privacy and security.

Cf. Section 6.1 for a means to synchronize private keys among different devices of the same network address in a secure manner.

4.2.2. Passphrase

Passphrases to protect a user's private key MUST be supported by pEp implementations, but SHOULD NOT be enforced by default. That is, if a pEp implementation finds a suitable (i.e., secure enough) cryptographic setup, which uses passphrases, pEp implementations MUST provide a way to unlock the key. However, if a new key pair is generated for a given identity, no passphrase SHOULD be put in place. The authors assume that the enforcement of secure (i.e., unique and long enough) passphrases would massively reduce the number of pEp users (by hassling them), while providing little to no additional privacy for the common cases of passive monitoring being carried out by corporations or state-level actors.

4.3. Public Key Distribution

As the pEp Key is available (cf. Section 4.1) implementers of pEp MUST ensure that by default the public key of the pEp Identity (cf. Section 3.4) in use (Own User; cf. Section 3.1.1) is attached to every outgoing message. However, this MAY be omitted if the pEp Identity in use (Own User) has previously received a message from the (remote) pEp Identity in question and that message was encrypted with public key that the would otherwise be attached.

The sender's public key SHOULD be sent encrypted whenever possible, i.e., when a public key of the receiving peer is available. If no encryption key of the recipient is available, the sender's public key MAY be sent unencrypted. In either case, this approach ensures that messaging clients (e.g., MUAs that at least implement OpenPGP) do not need to have pEp implemented to see a user's public key. Such peers thus have the chance to (automatically) import the sender's public key.

If there is already a known public key from the sender of a message and it is still valid and not expired, new keys MUST NOT be used for future communication unless they are signed by the previous key (to avoid a MITM attack). Messages MUST always be encrypted with the receiving peer's oldest public key, as long as it is valid and not expired.

4.3.1. UX Considerations

Implementers of pEp MUST prevent the display of public keys attached to messages (e.g, in email) to the user in order to prevent user confusion by files they are potentially unaware of how to handle.

4.3.2. No centralized public key storage or retrieval by default

Keyservers or generally intermediate approaches to obtain a peer's public key MUST NOT be used by default. Though, the user MAY be provided with an option (opt-in) to obtain keys from remote locations, in order to regard the widespread adoption of such approaches for key distribution.

Keys generated or obtained by pEp clients MUST NOT be uploaded to any (intermediate) keystore locations without the user's explicit consent.

4.3.3. Example message flow

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

The following example describes a pEp scenario between two users - Alice and Bob - in order to demonstrate the message flow that occurs when exchanging keys and determining basic trust management for the first time:

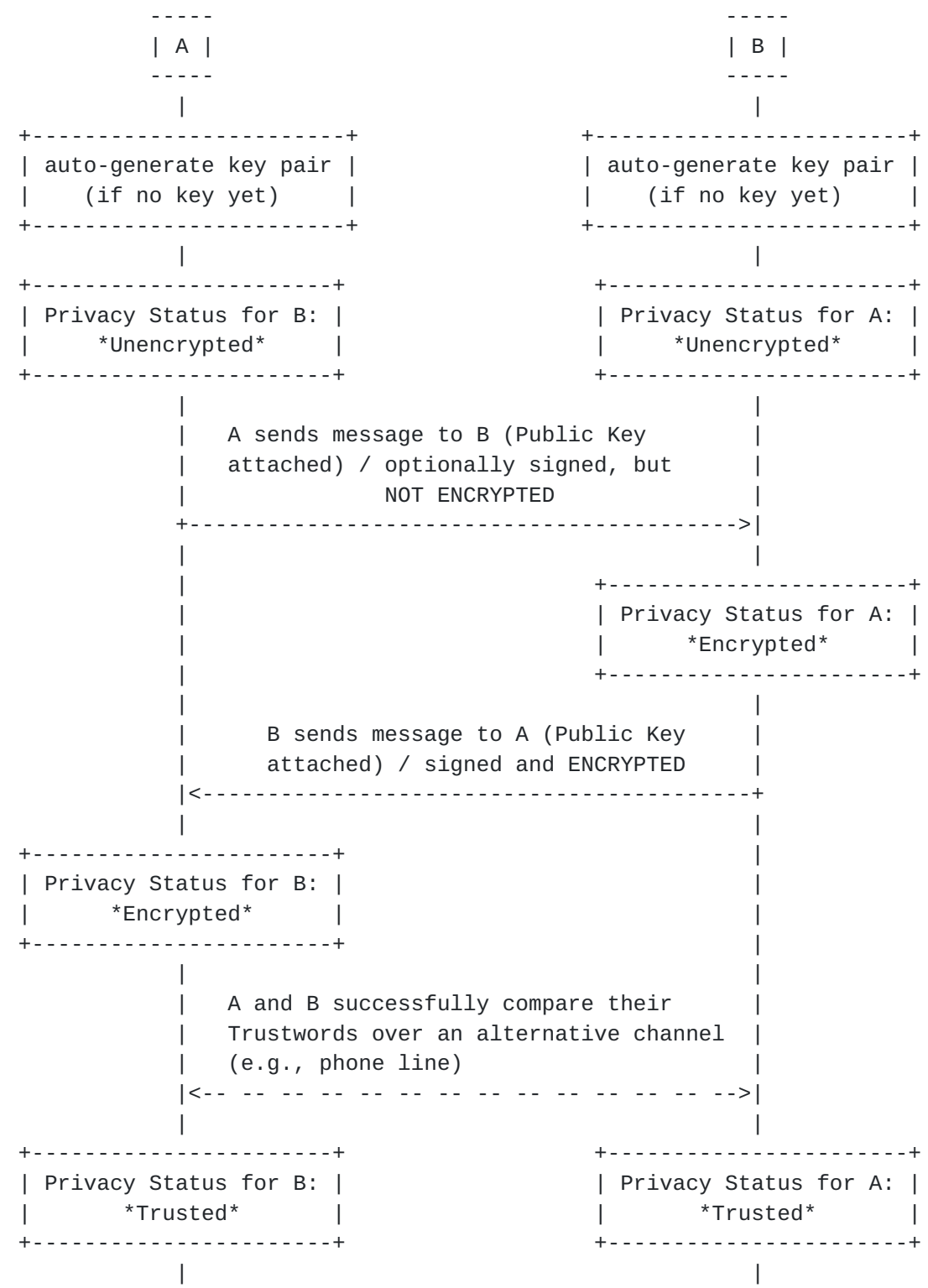
1. Alice - knowing nothing of Bob - sends a message to Bob. As Alice has no public key from Bob, this message is sent out unencrypted. However, Alice's public key is automatically attached.

2. Bob receives Alice's message and her public key. He is able to reply to her and encrypt the message. His public key is automatically attached to the message. Because he has her public key now, Alice's rating in his message client changes to 'encrypted'. From a UX perspective, this status is displayed in yellow (cf. Section 5.2).

3. Alice receives Bob's key. As of now Alice is also able to send secure messages to Bob. The rating for Bob changes to "encrypted" (with yellow color) in Alice's messaging client (cf. Section 5.2).

4. Alice receives Bob's reply with his public key attached. Now, Alice can send secure messages to Bob as well. The rating for Bob changes to yellow, or 'encrypted', in Alice's messaging client Section 5.2.

5. Alice and Bob can encrypt now, but they are not yet authenticated, leaving them vulnerable to man-in-the-middle (MitM) attacks. To prevent this from occurring, Alice and Bob can engage in a pEp Handshake to compare their Trustwords (cf. Section 5.3) and confirm if those match. After this step is performed, their respective identity ratings change to "encrypted and authenticated", which is represented by a green color (cf. Section 5.



4.4. Key Reset

Key Reset is specified in [I-D.pep-keyreset].

5. Trust Management

5.1. Privacy Status

The Privacy Status for an identity can change due to a number of factors. A change of privacy status will update the color code assigned to this identity accordingly, and is applied to future communications with this identity.

The Privacy Status is the most important component of pEp. Implementers of pEp MUST ensure that he Privacy Status is clearly visible to the user - on a per-recipient as well as on a per-message level, so that the user immediately understands from colors, symbols and texts how private

* a communication channel with a given peer was (or ought to be) and

* a given message was (or ought to be).

The Privacy Status is further specified in [I-D.pep-rating].

5.2. Trust Rating

pEp includes a Trust Rating system defining Rating and Color Codes to display the Privacy Status of a peer or message (cf. [I-D.pep-rating]):

* Ratings are labeled, e.g., as "Unencrypted", "Encrypted", "Trusted", "Under Attack", etc.

* The Privacy Status in its most general form is represented with traffic lights semantics (and respective symbols and words). The three colors "yellow", "green" and "red" are assigned to communication channels or messages (depending on the Privacy Status). Those immediately indicate how secure and trustworthy (and thus private) a communication was or ought to be considered. Note that there the may be no color applied, e.g. for the case that no public key is available to engage in private communications with an identity.

The pEp Trust Rating system with all its states and respective representations is specified in [I-D.pep-rating].

Note: An example for the rating of communication types, the definition of the data structure by the pEp Engine reference implementation is provided in Appendix A.2.

5.3. Handshake

To establish trust between peers and to upgrade the Privacy Status (cf. Section 5.1) of a peer, pEp defines a handshake, which is specified in [I-D.pep-handshake] and illustrated in Section 4.3.3 above. During the handshake Trustwords [I-D.pep-trustwords] are presented to pEp Users to compare and thus verify the authenticity of peers in order to mitigate MITM attacks. Trustwords are normally computed from the two peers' fingerprints of their public keys.

In order to retain compatibility with peers not using pEp implementations (e.g., Mail User Agents (MUAs) with OpenPGP [RFC4880] implementations without Trustwords), it is REQUIRED that pEp implementers provide the user with the option to show both peers' fingerprints instead of (or in addition to) the Trustwords.

6. Synchronization

An important feature of pEp is to assist the user to run pEp applications on different devices, such as personal computers, mobile phones and tablets, at the same time. Therefore, state needs to be synchronized among the different devices.

6.1. Private Key Synchronization

The pEp KeySync protocol (cf. [I-D.pep-keysync]) is a decentralized proposition which defines how pEp users can distribute their private keys among their different devices in a user-authenticated manner. This allows users to read their messages across their various devices, as long as they share a common channel, such as an email account.

7. Options in pEp

In this section a non-exhaustive selection of options is provided.

7.1. Option "Passive Mode"

By default, the sender attaches its public key to any outgoing message (cf. Section 4.3). For situations where a sender wants to ensure that it only attaches a public key to an Internet user which has a pEp implementation, a Passive Mode MUST be made available.

7.2. Option "Disable Protection"

Using this option, protection can be disabled generally or selectively. Implementers of pEp MUST provide an option "Disable Protection" to allow a user to disable encryption and signing for:

1. all communication

2. specific contacts

3. specific messages

Note that this option does not change the behavior on whether of not to attach the public key. Whether or not to attach the public key depends on the option "Passive Mode" (cf. Section 7.1).

7.3. Option "Extra Keys"

7.3.1. Use Case for Organizations

For internal or enterprise environments, authorized personnel may need to centrally decrypt user messages for archival or other legal purposes. Therefore, pEp implementers MAY provide an "Extra Keys" option in which a message is encrypted with at least one additional public key. The corresponding secret key(s) are intended to be secured by CISO staff or other authorized personnel for the organization.

However, it is crucial that the "Extra Keys" feature MUST NOT be activated by default for any network address, and is intended to be an option used only for organization-specific identities, as well as their corresponding network addresses and accounts. The "Extra Keys" feature SHOULD NOT be applied to the private identities, addresses, or accounts a user might possess once it is activated.

7.3.2. Use Case for Key Synchronization

The "Extra Keys" feature also plays a role during pEp's KeySync protocols [I-D.pep-keysync], where the additional keys are used to decipher message transactions by both parties involved during the negotiation process for private key synchronization. During the encrypted (but untrusted) transactions, KeySync messages are not just encrypted with the sending device's default key, but also with the default keys of both parties involved in the synchronization process.

7.4. Option "Blacklist Keys"

A "Blacklist Keys" option MAY be provided for an advanced user, allowing them to disable keys of peers that they no longer want to use in new communications. However, the keys MUST NOT be deleted. It MUST still be possible to verify and decipher past communications that used these keys.

7.5. Option "Trusted Server"

By default messages are stored encrypted on a server. With this option enabled messages will be stored unencrypted.

This may be useful in trusted environments (such as organizations) e.g., to conform to legal requirements such as 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 server regarded as trusted.

7.5.1. Changing Server Trust

Changing the server trust is possible at any time.

7.5.1.1. Changing from trusted to untrusted server

When changing from trusted server to untrusted server, implementations MAY encrypt all existing messages on the server, though this is NOT REQUIRED. However, new messages MUST be stored encrypted and whenever an existing (unencrypted) message is re-opened, it MUST be stored encrypted.

7.5.1.2. Changing from untrusted to trusted server

When changing from untrusted server to trusted server, implementation MAY decrypt all existing messages on the server, though this is NOT REQUIRED. However, new messages MUST be stored unencrypted and whenever an existing (encrypted) message is re-opened, it MUST be stored unencrypted.

8. Interoperability

pEp aims to be interoperable with existing applications designed to enable privacy, e.g., OpenPGP [RFC4880] and S/MIME [RFC8551] in email.

9. Security Considerations

By attaching the sender's public key to outgoing messages, Trust on First Use (TOFU) is established. However, this is prone to MITM attacks. Cryptographic key subversion is considered Pervasive Monitoring (PM) according to [RFC7258]. Those attacks can be mitigated, if the involved users compare their common Trustwords. This possibility MUST be made easily accessible to pEp users in the user interface implementation. If for compatibility reasons (e.g., with OpenPGP users) no Trustwords can be used, then a comparatively

easy way to verify the respective public key fingerprints MUST be implemented.

As the use of passphrases for private keys is not advised, devices themselves SHOULD use encryption.

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 an add-on for Thunderbird, release [SRC.pepforthunderbird]

Note: The former community project Enigmail/pEp as add-on for Thunderbird was discontinued and replaced by pEp's own add-on for Thunderbird [SRC.pepforthunderbird] in 2021.

pEp for Android, iOS, Outlook and Thunderbird are provided by pEp Security, a commercial entity specializing in end-user pEp implementations.

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

11.3. Reference implementation of pEp's core

The pEp Foundation provides a reference implementation of pEp's core principles and functionalities, which go beyond the documentation status of this Internet-Draft. [SRC.pepcore]

pEp's reference implementation is composed of pEp Engine and pEp Adapters (or bindings), alongside with some libraries which pEp Engine relies on to function on certain platforms (like a NetPGP fork we maintain for the iOS platform).

The pEp engine is a Free Software library encapsulating implementations of:

* Key Management

  Key Management in pEp engine is based on GnuPG key chains (NetPGP on iOS). Keys are stored in an OpenPGP compatible format and can be used for different crypto implementations.

* Trust Rating

  pEp engine is sporting a two phase trust rating system. In phase one there is a rating based on channel, crypto and key security named "comm_types". In phase 2 these are mapped to user representable values which have attached colors to present them in traffic light semantics.

* Abstract Crypto API

  The Abstract Crypto API is providing functions to encrypt and decrypt data or full messages without requiring an application programmer to understand the different formats and standards.

* Message Transports

  pEp engine will support a growing list of Message Transports to support any widespread text messaging system including email, SMS, XMPP and many more.

pEp engine is written in C99 programming language. It is not meant to be used in application code directly. Instead, pEp engine is coming together with a list of software adapters for a variety of programming languages and development environments, which are:

* pEp COM Server Adapter

* pEp JNI Adapter

* pEp JSON Adapter

* pEp ObjC (and Swift) Adapter

* pEp Python Adapter

11.4. Abstract Crypto API examples

A selection of code excerpts from the pEp Engine reference implementation (encrypt message, decrypt message, and obtain Trustwords) can be found in Appendix A.3.

12. Notes

The pEp logo and "pretty Easy privacy" are registered trademarks owned by the non-profit pEp Foundation based in Switzerland.

Primarily, we want to ensure the following:

* Software using the trademarks MUST be backdoor-free.

* Software using the trademarks MUST be accompanied by a serious (detailed) code audit carried out by a reputable third-party, for any proper release.

The pEp Foundation will help to support any community-run (non-commercial) project with the latter, be it organizationally or financially.

Through this, the foundation wants to make sure that software using the pEp trademarks is as safe as possible from a security and privacy point of view.

13. Acknowledgments

The authors would like to thank the following people who provided substantial contributions, helpful comments or suggestions for this document: Alexey Melnikov, Athena Schumacher, Ben Campbell, Brian Trammell, Bron Gondwana, Claudio Luck, Daniel Kahn Gillmor, Enrico Tomae, Eric Rescorla, Gabriele Lenzini, Hans-Peter Dittler, Iraklis Symeonidis, Kelly Bristol, Linda Carmen Schmid, Luca Saiu, Krista Bennett, Mirja Kuehlewind, Nana Karlstetter, Neal Walfield, Pete Resnick, Russ Housley, S. Shelburn, and Stephen Farrel.

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

14. References

14.1. Normative References

[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>.

[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>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

14.2. Informative References

[I-D.pep-email] Marques, H. and B. Hoeneisen, "pretty Easy privacy (pEp): Email Formats and Protocols", Work in Progress, Internet-Draft, draft-pep-email-02, 16 December 2022, <https://www.ietf.org/archive/id/draft-pep-email-02.txt>.

[I-D.pep-handshake]
Marques, H. and B. Hoeneisen, "pretty Easy privacy (pEp): Contact and Channel Authentication through Handshake", Work in Progress, Internet-Draft, draft-pep-handshake-00, 16 December 2022, <https://www.ietf.org/archive/id/draft-pep-handshake-00.txt>.

[I-D.pep-keyreset]
Hoeneisen, B., "pretty Easy privacy (pEp): Key Reset", Work in Progress, Internet-Draft, draft-pep-keyreset-00, 15 December 2022, <https://www.ietf.org/archive/id/draft-pep-keyreset-00.txt>.

[I-D.pep-keysync]
Birk, V., Hoeneisen, B., and K. Bristol, "pretty Easy privacy (pEp): Key Synchronization Protocol (KeySync)", Work in Progress, Internet-Draft, draft-pep-keysync-02, 13 July 2020, <https://www.ietf.org/archive/id/draft-pep-keysync-02.txt>.

[I-D.pep-rating]
Marques, H. and B. Hoeneisen, "pretty Easy privacy (pEp): Mapping of Privacy Rating", Work in Progress, Internet-Draft, draft-pep-rating-00, 16 December 2022, <https://www.ietf.org/archive/id/draft-pep-rating-00.txt>.

[I-D.pep-trustwords]
Hoeneisen, B. and H. Marques, "IANA Registration of Trustword Lists: Guide, Template and IANA Considerations", Work in Progress, Internet-Draft, draft-pep-trustwords-00, 16 December 2022, <https://www.ietf.org/archive/id/draft-pep-trustwords-00.txt>.

[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/>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[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>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

[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>.

[RFC8280] ten Oever, N. and C. Cath, "Research into Human Rights Protocol Considerations", RFC 8280, DOI 10.17487/RFC8280, October 2017, <https://www.rfc-editor.org/info/rfc8280>.

[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.pepcore] "Core source code and reference implementation of pEp (engine and adapters)", March 2022, <https://gitea.pep.foundation/pEp.foundation/pEpEngine>.

[SRC.pepforandroid] "Source code for pEp for Android", December 2022, <https://pep-security.lu/gitlab/android/pep>.

[SRC.pepforios] "Source code for pEp for iOS", December 2022, <https://pep-security.lu/gitlab/iOS/pep4ios>.

[SRC.pepforoutlook] "Source code for pEp for Outlook", December 2022, <https://pep-security.lu/gitlab/win/pEpForOutlook>.

[SRC.pepforthunderbird]
"Source code for pEp for Thunderbird", December 2022,
<https://pep-security.lu/gitlab/thunderbird/pEpForThunderbird>.

Appendix A. Code Excerpts

This section provides excerpts of the running code from the pEp reference implementation pEp engine (C99 programming language).

A.1. pEp Identity

How the pEp Identity is defined in the data structure (cf. src/pEpEngine.h):

```
typedef struct _pEp_identity {
    char *address;             ///< C string with address UTF-8 encoded
    char *fpr;                 ///< C string with fingerprint UTF-8
                               ///< encoded
    char *user_id;             ///< C string with user ID UTF-8
                               ///< encoded\n
                               ///< user_id MIGHT be set to
                               ///< "pEp_own_userId" (use PEP_OWN_USERID
                               ///< preprocessor define)
                               ///< if this is own user's identity.
                               ///< But it is not REQUIRED to be.
    char *username;            ///< C string with user name UTF-8
                               ///> encoded
    PEP_comm_type comm_type;   ///< type of communication with this ID
    char lang[3];              ///< language of conversation
                               ///< ISO 639-1 ALPHA-2, last byte is 0
    bool me;                   ///< if this is the local user
                               ///< herself/himself
    unsigned int major_ver;    ///< highest version of pEp message
                               ///< received, if any
    unsigned int minor_ver;    ///< highest version of pEp message
                               ///< received, if any
    PEP_enc_format enc_format; ///< Last specified format we encrypted
                               ///< to for this identity
    identity_flags_t flags;    ///< identity_flag1 | identity_flag2
                               ///< | ...
} pEp_identity;
```

A.1.1. Corresponding SQL

Relational table scheme excerpts (in SQL) used in current pEp implementations, held locally for every pEp installation in a SQLite database:

```
   CREATE TABLE person (
      id text primary key,
      username text not null,
      main_key_id text
          references pgp_keypair (fpr)
          on delete set null,
      lang text,
      comment text,
      is_pEp_user integer default 0
   );

   CREATE TABLE identity (
      address text,
      user_id text
          references person (id)
          on delete cascade on update cascade,
      main_key_id text
          references pgp_keypair (fpr)
          on delete set null,
      comment text,
      flags integer default 0,
      is_own integer default 0,
      pEp_version_major integer default 0,
      pEp_version_minor integer default 0,
      enc_format integer default 0,
      timestamp integer default (datetime('now')),
      primary key (address, user_id)
   );

   CREATE TABLE pgp_keypair (
      fpr text primary key,
      created integer,
      expires integer,
      comment text,
      flags integer default 0
   );
```

A.2. pEp Communication Type

In the following, is an example for the rating of communication types as defined by a data structure (cf. src/pEpEngine.h [SRC.pepcore]):

```
typedef enum _PEP_comm_type {
    PEP_ct_unknown = 0,

    // range 0x01 to 0x09: no encryption, 0x0a to 0x0e: nothing
    // reasonable
```

```
    PEP_ct_no_encryption = 0x01,                // generic
    PEP_ct_no_encrypted_channel = 0x02,
    PEP_ct_key_not_found = 0x03,
    PEP_ct_key_expired = 0x04,
    PEP_ct_key_revoked = 0x05,
    PEP_ct_key_b0rken = 0x06,
    PEP_ct_key_expired_but_confirmed = 0x07, // NOT with confirmed bit.
                                             // Just retaining info here
                                             // in case of renewal.
    PEP_ct_my_key_not_included = 0x09,

    PEP_ct_security_by_obscurity = 0x0a,
    PEP_ct_b0rken_crypto = 0x0b,
    PEP_ct_key_too_short = 0x0c,

    PEP_ct_compromised = 0x0e,        // known compromised connection
    PEP_ct_compromized = 0x0e,        // deprecated misspelling
    PEP_ct_mistrusted = 0x0f,         // known mistrusted key

    // range 0x10 to 0x3f: unconfirmed encryption

    PEP_ct_unconfirmed_encryption = 0x10,       // generic
    PEP_ct_OpenPGP_weak_unconfirmed = 0x11,     // RSA 1024 is weak

    PEP_ct_to_be_checked = 0x20,                // generic
    PEP_ct_SMIME_unconfirmed = 0x21,
    PEP_ct_CMS_unconfirmed = 0x22,

    PEP_ct_strong_but_unconfirmed = 0x30,       // generic
    PEP_ct_OpenPGP_unconfirmed = 0x38,          // key at least 2048 bit
                                                // RSA or EC
    PEP_ct_OTR_unconfirmed = 0x3a,

    // range 0x40 to 0x7f: unconfirmed encryption and anonymization

    PEP_ct_unconfirmed_enc_anon = 0x40,         // generic
    PEP_ct_pEp_unconfirmed = 0x7f,

    PEP_ct_confirmed = 0x80,                    // this bit decides if
                                                // trust is confirmed

    // range 0x81 to 0x8f: reserved
    // range 0x90 to 0xbf: confirmed encryption

    PEP_ct_confirmed_encryption = 0x90,         // generic
    PEP_ct_OpenPGP_weak = 0x91,                 // RSA 1024 is weak
                                                // (unused)
```

```
    PEP_ct_to_be_checked_confirmed = 0xa0,      // generic
    PEP_ct_SMIME = 0xa1,
    PEP_ct_CMS = 0xa2,

    PEP_ct_strong_encryption = 0xb0,            // generic
    PEP_ct_OpenPGP = 0xb8,                      // key at least 2048 bit
                                                // RSA or EC
    PEP_ct_OTR = 0xba,

    // range 0xc0 to 0xff: confirmed encryption and anonymization

    PEP_ct_confirmed_enc_anon = 0xc0,           // generic
    PEP_ct_pEp = 0xff
} PEP_comm_type;
```

A.3. Abstract Crypto API examples

The following code excerpts are from the pEp Engine reference implementation, to be found in src/message_api.h.

[[Note: Just a selection; more functionality is available.]]

A.3.1. Encrypting a Message

Cf. src/message_api.h [SRC.pepcore]:

```
/**
 *  <!--       encrypt_message()       -->
 *
 *  @brief Encrypt message in memory
 *
 *  @param[in]     session       session handle
 *  @param[in,out] src           message to encrypt - usually in-only
 *                               except for the rating field, but can be
 *                               in-out for unencrypted messages;
 *                               in that case, we may attach the key
 *                               and decorate the message.
 *                               In any case, reset the rating.
 *  @param[in]     extra         extra keys for encryption
 *  @param[out]    dst           pointer to new encrypted message or
 *                               NULL if no
 *                               encryption could take place
 *  @param[in]     enc_format    The desired format this message should
 *                               be encrypted with
 *  @param[in]     flags         flags to set special encryption
 *                               features
 *
 *  @retval PEP_STATUS_OK            on success
```

```
 *  @retval PEP_KEY_HAS_AMBIG_NAME    at least one of the receipient
 *                                    keys has an ambiguous name
 *  @retval PEP_UNENCRYPTED           on demand or no recipients with
 *                                    usable key, is left unencrypted,
 *                                    and key is attached to it
 *  @retval PEP_ILLEGAL_VALUE         illegal parameter values
 *  @retval PEP_OUT_OF_MEMORY         out of memory
 *  @retval any other value on error
 *
 *  @warning the ownership of src remains with the caller
 *           the ownership of dst goes to the caller
 *
 *  @warning enc_format PEP_enc_inline_EA:
 *         internal format of the encrypted attachments is changing, see
 *         https://dev.pep.foundation/Engine/ElevatedAttachments\n
 *         Only use this for transports without support for attachments
 *         when attached data must be sent inline
 *
 */

DYNAMIC_API PEP_STATUS encrypt_message(
        PEP_SESSION session,
        message *src,
        stringlist_t *extra,
        message **dst,
        PEP_enc_format enc_format,
        PEP_encrypt_flags_t flags
    );
```

A.3.2. Decrypting a Message

Cf. src/message_api.h [SRC.pepcore]:

```
/**
 *  <!--       decrypt_message_2()     -->
 *
 *  @brief Decrypt message in memory
 *
 *  @param[in]     session   session handle
 *  @param[in,out] src       message to decrypt - see warning about
 *                           identities below.
 *                           the rating field of src (instead of dst)
 *                           is updated in case encryption fails.
 *  @param[out]    dst       pointer to new decrypted message or NULL
 *                           on failure
 *  @param[in,out] keylist   in: stringlist with additional keyids
 *                           for reencryption if needed (will be freed
 *                           and replaced with output keylist)
```

```
 *                          out: stringlist with keyids used for
 *                          signing and encryption. first key is
 *                          signer, additional keys are the ones it was
 *                          encrypted to. Only signer and whichever
 *                          of the user's keys was used are reliable
 *  @param[in,out] flags    flags to signal special decryption features
 *
 *  @retval <ERROR>               any error status
 *  @retval PEP_DECRYPTED         if message decrypted but not verified
 *  @retval PEP_CANNOT_REENCRYPT  if message was decrypted (and possibly
 *                                verified) but a reencryption operation
 *                                is expected by the caller and failed
 *  @retval PEP_STATUS_OK         on success
 *
 *  @note Flags above are as follows:
 *  @verbatim
 *  ---------------------------------------------------------------------|
 *  Incoming flags                                                       |
 *  ---------------------------------------------------------------------|
 *  Flag                               | Description                     |
 *  -----------------------------------|---------------------------------|
 *  PEP_decrypt_flag_untrusted_server  | used to signal that decrypt     |
 *                                     | function should engage          |
 *                                     | in behaviour specified for      |
 *                                     | when the server storing         |
 *                                     | the source is untrusted.        |
 *  ---------------------------------------------------------------------|
 *  Outgoing flags                                                       |
 *  ---------------------------------------------------------------------|
 *  PEP_decrypt_flag_own_private_key   | private key was imported for    |
 *                                     |  one of our addresses           |
 *                                     | (NOT trusted or set to be used  |
 *                                     | - handshake/trust is required   |
 *                                     | for that)                       |
 *                                     |                                 |
 *  PEP_decrypt_flag_src_modified      | indicates that the              |
 *                                     | modified_src field should       |
 *                                     | contain a modified version of   |
 *                                     | the source, at the moment       |
 *                                     | always as a result of the       |
 *                                     | input flags.                    |
 *                                     |                                 |
 *  PEP_decrypt_flag_consume           | used by sync to indicate this   |
 *                                     | was a pEp internal message and  |
 *                                     | should be consumed externally   |
 *                                     | withoutshowing it as a normal   |
 *                                     | message to the user             |
 *                                     |                                 |
```

```
 *  PEP_decrypt_flag_ignore           | used by sync                   |
 *  ----------------------------------------------------------------|
 *  @endverbatim
 *
 *  @ownership src remains with the caller; HOWEVER, the contents
 *               might be modified (strings freed and allocated anew or
 *               set to NULL, etc) intentionally; when this happens,
 *               PEP_decrypt_flag_src_modified is set.
 *
 *  @ownership dst goes to the caller
 *
 *  @ownership contents of keylist goes to the caller
 *
 *  @note if src is unencrypted this function returns PEP_UNENCRYPTED
 *        and sets dst to NULL
 *  @note if src->enc_format is PEP_enc_inline_EA on input then elevate
 *        attachments will be expected
 *
 *
 *  @warning decrypt_message RELIES on the fact that identity
 *           information provided in src for recips and sender is AS
 *           TAKEN FROM THE ORIGINAL PARSED MESSAGE. This means that if
 *           update_identity or myself is called on those identities by
 *           the caller before passing the message struct to
 *           decrypt_message, the caller will have to cache and restore
 *           those to their original state before sending them to this
 *           function.
 *           ADAPTERS AND APPLICATIONS PLEASE TAKE NOTE OF THIS.
 *           (Obviously, this doesn't  include information like
 *           user_ids, but we very specifically need the incoming
 *           usernames preserved so that they can be handled by the
 *           internal algorithm appropriately)
 */

DYNAMIC_API PEP_STATUS decrypt_message_2(
        PEP_SESSION session,
        message *src,
        message **dst,
        stringlist_t **keylist,
        PEP_decrypt_flags_t *flags
);
```

A.3.3. Obtain Common Trustwords

Cf. src/message_api.h [SRC.pepcore]:

```
/**
 *  <!--       get_trustwords()       -->
 *
 *  @brief Get full trustwords string for a *pair* of identities
 *
 *  @param[in]   session    session handle
 *  @param[in]   id1        identity of first party in communication -
 *                          fpr can't be NULL
 *  @param[in]   id2        identity of second party in communication -
 *                          fpr can't be NULL
 *  @param[in]   lang       C string with ISO 639-1 language code
 *  @param[out]  words      pointer to C string with all trustwords
 *                          UTF-8 encoded, separated by a blank each
 *                          NULL if language is not supported or
 *                          trustword wordlist is damaged or unavailable
 *  @param[out]  wsize      length of full trustwords string
 *  @param[in]   full       if true, generate ALL trustwords for these
 *                          identities.
 *                          else, generate a fixed-size subset.
 *                          (TODO: fixed-minimum-entropy
 *                          subset in next version)
 *
 *  @retval PEP_STATUS_OK            trustwords retrieved
 *  @retval PEP_OUT_OF_MEMORY        out of memory
 *  @retval PEP_ILLEGAL_VALUE        illegal parameter values
 *  @retval PEP_TRUSTWORD_NOT_FOUND  at least one trustword not found
 *
 *  @warning the word pointer goes to the ownership of the caller.
 *           the caller is responsible to free() it (on Windoze use
 *           pEp_free())
 *
 */
DYNAMIC_API PEP_STATUS get_trustwords(
        PEP_SESSION session, const pEp_identity* id1,
        const pEp_identity* id2, const char* lang, char **words,
        size_t *wsize, bool full
    );
```

Appendix B. Document Changelog

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

* draft-pep-general-02:

  - Reorganized chapter structure to align with [I-D.pep-email]

  - Update Terms and References

- Serveral corrections and enhancements

* draft-pep-general-01:
  - Minor update in Section 2.1
  - Rewrite Section 4.2.1
* draft-pep-general-00:
  - Major Rewrite of "pEp Identity Concept" section
  - Major Rewrite of "Key Management" section
  - Update Code Excerpts to match current implementation state
  - Lots of corrections and editorial changes
* draft-birk-pep-06:
  - Minor changes
* draft-birk-pep-05:
  - Minor changes, especially in identity system
* draft-birk-pep-04:
  - Fix internal reference
  - Add IANA Considerations section
  - Add other use case of Extra Keys
  - Incorporate review changes by Kelly Bristol and Nana Karlstetter
* draft-birk-pep-03:
  - Major restructure of the document
  - Added several new sections, e.g., Key Reset, Trust Revoke, Trust Synchronization, Private Key Export / Import, Privacy Considerations (content yet mostly TODO)
  - Added reference to HRPC work / RFC8280

    o Added text and figure to better explain pEp's automated Key Exchange and Trust management (basic message flow)

  - Lots of improvement in text and editorial changes

* draft-birk-pep-02:

  - Move (updated) code to Appendix

  - Add Changelog to Appendix

  - Add Open Issue section to Appendix

  - Fix description of what Extra Keys are

  - Fix Passive Mode description

  - Better explain pEp's identity system

* draft-birk-pep-01:

  - Mostly editorial

* draft-birk-pep-00:

  - Initial version

Appendix C. Open Issues

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

* Shorten Introduction and Abstract

Authors' Addresses

Volker Birk
pEp Foundation
Oberer Graben 4
CH- 8400 Winterthur
Switzerland
Email: volker.birk@pep.foundation
URI: https://pep.foundation/

Hernani Marques
pEp Foundation
Oberer Graben 4
CH- 8400 Winterthur
Switzerland
Email: hernani.marques@pep.foundation
URI: https://pep.foundation/

Bernie Hoeneisen
pEp Foundation
Oberer Graben 4
CH- 8400 Winterthur
Switzerland
Email: bernie.hoeneisen@pep.foundation
URI: https://pep.foundation/