Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 13, 2018

V. Birk
H. Marques
Shelburn
pEp Foundation
S. Koechli
pEp Security

January 09, 2018

pretty Easy privacy (pEp): Privacy by Default
draft-birk-pep-01

Abstract

Building on already available security formats and message transports (like PGP/MIME for email), and with the intention to stay interoperable to systems widespreadly deployed, pretty Easy privacy (pEp) describes protocols to automatize operations (key management, key discovery, private key handling including peer-to-peer synchronization of private keys and other user data across devices) that have been seen to be barriers to deployment of end-to-end secure interpersonal messaging. pEp also introduces "Trustwords" (instead of fingerprints) to verify communication 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. In this document, the general design choices and principles of pEp are outlined.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[[At this stage it is not year clear to us how many of our implementation details should be part of new RFCs and at which places we can safely refer to already existing RFCs to make clear on which RFCs we are already relying.]]

The pretty Easy privacy (pEp) protocols are propositions to the Internet community to create software for peers to automatically encrypt, anonymize (where possible, depending on the message transport used) and verify their daily written digital communications -- this is done by building upon already existing standards and tools and automatizing all steps a user would need to carry out to engage in secure end-to-end encrypted communciations without depending on centralized infrastructures.

To mitigiate for Man-In-The-Middle Attacks (MITM) and as the only manual step users may carry out, Trustwords as natural language representations of two peers' fingerprints are proposed, for peers to put trust on their communication channel.

Particularly, pEp proposes to automatize key management, key discovery and also synchronization of secret key material by an in-band peer-to-peer approach.

[[The pEp initiators had to learn from the CryptoParty movement, from which the project emerged, that step-by-step guides can be helpful to a particiular set of users to engage in secure end-to-end communications, but that for a much major fraction of users it would be more convenient to have the step-by-step procedures put into actual code (as such, following a protocol) and thus automatizing the initial configuration and whole usage of cryptographic tools.]]

The Privacy by Default principles that pretty Easy privacy (pEp) introduces, are in accordance with the perspective outlined in [RFC7435] to bring Opportunistic Security in the sense of "some protection most of the time", with the subtle, but important difference that when privacy is weighted against security, the choice falls to privacy, which is why in pEp data minimization is a primary goal (e.g., omitting unnecessary email headers or encrypting the subject line).

The pEp propositions are focused on written digital communications, but not limited to asynchronous (offline) types of communications like email, but can also be implemented for message transports, which support synchronous (online) communications (e.g., for peer-to-peer networks like GNUnet). pEp's goal is to bridge the different standardized and/or widely spread communications channels, such that users can reach their peers in the most privacy-enhancing way possible using differing IRIs/URIs.

2. Terms

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].

Terms like "TOFU" or "MITM" are used as defined in [RFC4949].

- Handshake: The process when Alice -- e.g. in-person or via phone -- contacts Bob to verifiy Trustwords (or by fallback: fingerprints) is called 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. [pEpTrustwords]

3. Protocol's core design principles

3.1. Compatibility

- Be conservative (strict) in requirements for pEp implementations and how they behave between each other.

- Be liberal (accepting) in what comes in from non-pEp implementations (e.g., do not send, but support to decipher PGP/INLINE formats).

- Where pEp requires diverging from an RFC for privacy reasons (e.g., from OpenPGP propositions as defined in [RFC4880]), options SHOULD be implemented to empower the user to comply to practices already (widespreadly) used, either at contact level or globally.

3.2. Peer-to-Peer (P2P)

All communications and verifications in pEp implementations for pursuing secure and establishing trusted communications between peers MUST be Peer-to-Peer (P2P) in nature.

This means, there SHALL NOT be any pEp-specific central services whatsoever needed for implementers of pEp to rely on, neither for verification of peers nor for the actual encryption.

Still, implementers of pEp MAY provide options to interoperate with providers of centralized infrastructures as users MAY NOT be stopped from being able to communicate with their peers on platforms with vendor lock-in.

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

3.3. User Experience (UX)

[[We are aware of the fact that usually UX requirements are not part of RFCs. However, to have massively more people engaged in secure end-to-end encryption and at the same time to avoid putting users at risk, we believe requiring certain straightforward signaling for the users to be a good idea -- in a similar way as this happens to be the case for already popular Instant Messaging services.]]

Implementers of pEp MUST take special care to not confuse users with t echnical terms, especially those of cryptography (e.g., "keys", "certificates" or "fingerprints") if they do not explicitly ask for them. Advanced settings MAY be available, in some cases further options MUST be available, but they SHALL NOT be unnecessarily exposed to users of pEp implementations at the first sight when using clients implementing the pEp propositions.

The authors believe widespread adoption of end-to-end cryptography is much less of an issue if the users are not hassled and visibly forced in any way to use cryptography. That is, if they can just rely on the principles of Privacy by Default.

By consequence, this means that users MUST NOT wait for cryptographic tasks (e.g., key generation or public key retrieval) to finish before being able to have their respective message clients ready to communicate. This finally means, pEp implementers MUST make sure that the ability to draft, send and receive messages is always preserved -- even if that means a message is sent out unencrypted, thus being in accordance with the Opportunistic Security approach outlined in [RFC7435].

In turn, pEp implementers MUST make sure a Privacy Status is clearly visible to the user on both contact and message level, such that

users easily understand with what level of privacy messages are about to be sent or were received, respectively.

4. Identities in pEp

With pEp users MUST have the possibility to have different identities, which MUST not correlate to each other by default. On the other hand, binding of different identities MUST be supported (being for support of aliases).

[[This is the reason why in current pEp implementations for each email account a different key pair is created, which allows a user to retain different identities.]]

In particular, with pEp users MUST NOT be bound to a specific IRI/URI, but SHALL be free to choose which identity they want to expose to certain peers -- this includes support for pseudonymity and anonymity, which the authors consider to be vital for users to control their privacy.

[[It might be necessary to introduce further addressing schemes through IETF contributions or IANA registrations.]]

In the reference implementation of the pEp Engine (cf. src/pEpEngine.h), a pEp identity is defined like the following (C99):

```
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
    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
    identity_flags_t flags;     // identity_flag1 | identity_flag2 | ...
} pEp_identity;
```

A relational example (in SQL) used in current pEp implementations:

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

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,
    device_group text
);

CREATE TABLE identity (
    address text,
    user_id text
        references person (id)
        on delete cascade,
    main_key_id text
        references pgp_keypair (fpr)
        on delete set null,
    comment text,
    flags integer default 0,   primary key (address, user_id)
 );
```

The public key's fingerprint (denoted as fpr) as part of a pEp identity MUST always be the full fingerprint.

Notable differences of how terms and concepts used differ between pEp and OpenPGP:

pEp	OpenPGP	Comments
user_id	(no concept)	ID for a person, i.e. a contact
username + address	uid	comparable only for email
fpr	fingerprint	used as key ID in pEp
(no concept)	Key ID	does not exist in pEp

5. Key Management

Key management in pEp MUST be automatized in order to achieve the goal of widespread adoption of secure communications.

A pEp implementation MUST make sure cryptographic keys for end-to-end cryptography are generated for every identity configured (or instantly upon its configuration) if no secure cryptographic setup can be found. Users SHALL NOT be stopped from communicating -- this also applies for initial situations where cryptographic keys are not generated fast enough. This process MUST be carried out in the background so the user is not stopped from communicating.

There is the pEp Trust Rating system in [pEpTrustRating] describing which kind of encryption MUST be considered reliable and is thus secure enough for usage in pEp implementations. This also applies for keys already available for the given identity. If the available keys are considered unsecure (e.g, insufficent key length), pEp implementers are REQUIRED to generate new keys for use with the respective identity.

As example for the rating of communication types, the definition of the data structure by the pEp Engine reference implementation (cf. src/pEpEngine.h) is provided:

```
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_my_key_not_included = 0x09,

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

    PEP_ct_compromized = 0x0e, // known compromized connection
    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;
```

5.1. Private Keys

Private keys in pEp implementations MUST always be held on the end user's device(s): pEp implementers SHALL NOT rely on private keys stored in centralized remote locations. This also applies for key storages where the private keys are protected with sufficiently long passphrases. It MUST be considered a violation of pEp's P2P design principle to rely on centralized infrastructures. 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 done in a way the locally-held private key can neither be directly accessed nor leaked to the outside world.

[[It is particularly important that browser add-ons implementing pEp functionality do not obtain their cryptographic code from a centralized (cloud) service, as this must be considered a centralized attack vector allowing for backdoors, negatively impacting privacy.]]

As a decentralized proposition, there is a pEp Key Synchronization protocol. [pEpKeySync] It outlines how pEp implementers can distribute their private keys in a secure and trusted manner: this allows Internet users to read their messages across their different devices, when sharing a common address (e.g., the same email account).

5.2. Key Distribution

Implementers of pEp are REQUIRED to attach the identity's public key to any outgoing message. However, this MAY be ommitted if you previously received a message encrypted with the public key of the receiver.

The sender's public key MUST be sent encrypted whenever possible, i.e. when a public key of the receiving peer is available.

If no encryption key is available for the recipient, the sender's public key MUST be sent unencrypted. In either case, this approach ensures that message clients (e.g., MUAs which 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 SHALL not be used for future communication to avoid a MITM attack unless they are signed by the previous key. Messages SHALL always be encrypted with the receiving peer's oldest public key, as long as it is valid and not expired.

Implementers of pEp SHALL make sure that public keys attached to messages (e.g, in email) are not displayed to the user. This ensures, they do not get confused by a file they cannot potentially deal with.

Metadata (e.g., email headers) SHALL NOT be made to announce a user's public key by the Privacy by Default principles. This must be considered unnecessary information leakage, potentially affecting privacy -- also depending on a country's data retention laws.

Furtherly, this affects interoperability to existing users (e.g., in the OpenPGP field) which have no notion of such header fields and thus lose the ability to import any such keys distributed this way. It SHOULD, though, be supported to obtain other users' public keys by extracting them from respective header fields, in case such approaches get widespread.

Keyservers or generally intermediate approaches to obtain a peer's public key SHALL NOT be used by default. On the other hand, the user MAY be given the option to opt-in for remote locations to obtain keys, considering the widespread adoption of such approaches for key distribution.

Keys generated or obtained by implementations SHALL NOT be uploaded to any (intermediate) keystore locations without the user's explicit will.

5.3. Passphrases

Passphrases to protect an user's private key MUST be supported by pEp implementations, but SHALL NOT be enforced by default. That is, if a pEp implementation finds a suitable (i.e., secure enough) crytographic 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 SHALL be put in place. The authors assume that the enforcement of secure (i.e., unique and long enough) passphrases would massively reduce the users of pEp (by hassling them) -- and in turn provide little to no additional privacy for the common cases of passive monitoring being carried out by corporations and state-level actors.

6. Privacy Status

For end-users, the most important component of pEp, which MUST be made visible on a per-recipient and per-message level, is the Privacy Status.

By colors, symbols and texts a user SHALL immediately understand 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 in its most general form MUST be expressed with traffic lights semantics (and respective symbols and texts), whereas the three colors yellow, green and red can be applied for any peer or message -- like this immediately indicating how secure and

trustworthy (and thus private) a communication was or ought to be considered. In cases no (special) Privacy Status can be inferred for peers or messages, no color (or the gray color) MUST be shown and respective texts -- being "unknown" or "unreliable" -- MUST be shown.

The detailed Privacy Status as an end-user element of the pEp Trust Rating system with all its states and respective represenations to be followed is outlined in [pEpTrustRating].

7. Options in pEp

[[Just a selection; not yet complete.]]

7.1. Option "Passive Mode"

For situations where it might not be desirable to attach the sender's public key for outgoing messages (which is the default), a "Passive Mode" option MUST be made availble to avoid this.

7.2. Option "Disable Protection"

7.2.1. For all communications

Implementers of pEp MUST provide an option "Disable Protection" for the user's will to disable any outgoing encryption and signing. This option SHALL not affect the user's ability to decipher already received or sent messages.

7.2.2. For some communications

For users to disable protection for some situations, i.e. at contact or message level, pEp implementers MUST provide an option. This allows users to disable outgoing encryption and signing for peers or individual messages.

7.3. Option "Extra Keys"

For environments where there is the need to send messages to further locations, pEp implementers MAY provide an "Extra Keys" option where further recipients (by public key) can be specified. With the user's will, each outgoing message MUST then be sent encrypted to any of those extra (implicit) recipients. Message clients SHOULD save and show only one message as sent to the explicit recipient(s), so as to not confuse users.

7.4. Option "Blacklist Keys"

An option "Blacklist Keys" MUST be provided for an advanced user to be able to disable keys which the user does not want to be used anymore for any new communications. However, the keys SHALL NOT be deleted. It MUST still be possible to verify and decipher past communications.

7.5. Establishing trust between peers

In pEp, Trustwords [pEpTrustwords] are used for users to compare the authenticity of peers in order to mitigate for MITM attacks.

By default, Trustwords MUST be used to represent two peers' fingerprints of their public keys in pEp implementations.

In order to retain compatibility with peers not using pEp implementations (e.g., Mail User Agents (MUAs) with OpenPGP implementations without Trustwords), it is REQUIRED that pEp implementers give the user the choice to show both peers' fingerprints instead of just their common Trustwords.

8. Security Considerations

By attaching the sender's public key to outgoing messages, Trust on First Use (TOFU) is established, which can lead for MITM attacks to succeed. Cryptographic key subversion is considered Pervasive Monitoring (PM) according to [RFC7258]. Those attacks can be mitigated by having the involved users comparing 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 an comparibly easy way to verify the respective public key fingerprints MUST be implemented.

Devices themselves SHOULD be made encrypted, as the use of passphrases for private keys is not advised.

9. Implementation Status

9.1. 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".

9.2. 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. [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 compatbile 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. 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
- pEp Qt Adapter

9.3. Abstract Crypto API examples

[[Just a selection; more functionality available.]]

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

9.3.1. Encrypting a message

```
  // encrypt_message() - encrypt message in memory
  //
  //  parameters:
  //      session (in)     session handle
  //      src (in)         message to encrypt
  //      extra (in)       extra keys for encryption
  //      dst (out)        pointer to new encrypted message or NULL on failure
  //      enc_format (in)  encrypted format
  //      flags (in)       flags to set special encryption features
  //
  //  return value:
  //      PEP_STATUS_OK           on success
  //      PEP_KEY_NOT_FOUND       at least one of the receipient keys
  //                              could not be found
  //      PEP_KEY_HAS_AMBIG_NAME  at least one of the receipient keys has
  //                              an ambiguous name
  //      PEP_GET_KEY_FAILED      cannot retrieve key
  //      PEP_UNENCRYPTED         no recipients with usable key,
  //                              message is left unencrypted,
  //                              and key is attached to it
  //
  //  caveat:
  //      the ownershop of src remains with the caller
  //      the ownership of dst goes to the caller
  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
      );
```

9.3.2. Decrypting a message

```
  // decrypt_message() - decrypt message in memory
  //
  //  parameters:
  //      session (in)   session handle
  //      src (in)       message to decrypt
  //      dst (out)      pointer to new decrypted message or NULL on failure
  //      keylist (out)  stringlist with keyids
  //      rating (out)   rating for the message
  //      flags (out)    flags to signal special decryption features
  //
  //  return value:
  //      error status
  //      or PEP_DECRYPTED if message decrypted but not verified
  //      or PEP_STATUS_OK on success
  //
  // caveat:
  //      the ownership of src remains with the caller
  //      the ownership of dst goes to the caller
  //      the ownership of keylist goes to the caller
  //      if src is unencrypted this function returns PEP_UNENCRYPTED and sets
  //      dst to NULL
  DYNAMIC_API PEP_STATUS decrypt_message(
          PEP_SESSION session,
          message *src,
          message **dst,
          stringlist_t **keylist,
          PEP_rating *rating,
          PEP_decrypt_flags_t *flags
  );
```

9.3.3. Obtaining common Trustwords

```
  // get_trustwords() - get full trustwords string for a *pair* of identities
  //
  //    parameters:
  //        session (in)  session handle
  //        id1 (in)      identity of first party in communication - fpr can't
be NULL
  //        id2 (in)      identity of second party in communication - fpr can't
be NULL
  //        lang (in)     C string with ISO 639-1 language code
  //        words (out)   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
  //        wsize (out)   length of full trustwords string
  //        full (in)     if true, generate ALL trustwords for these
identities.
  //                      else, generate a fixed-size subset. (TODO: fixed-
minimum-entropy
  //                       subset in next version)
  //
  //    return value:
  //        PEP_STATUS_OK            trustwords retrieved
  //        PEP_OUT_OF_MEMORY        out of memory
  //        PEP_TRUSTWORD_NOT_FOUND  at least one trustword not found
  //
  //    caveat:
  //        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
  );
```

9.4. Current software implementing pEp

The following software implementing the pEp protocols (to varying degrees) already exists; it does not yet go beyond implementing pEp for email, which is to be described nearer in [pEpEmail]:

- pEp for Outlook as addon for Microsoft Outlook, production [pEpForOutlookSrc]
- pEp for Android (based on a fork of the K9 MUA), beta [pEpForAndroidSrc]
- Enigmail/pEp as addon for Mozilla Thunderbird, early beta

[EnigmailpEpSrc]

- pEp for iOS (implemented in a new MUA), alpha [pEpForiOSSrc]

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

10. Notes

The pEp logo and "pretty Easy privacy" are registered trademarks owned by pEp Foundation in Switzerland, a tax-free, non-commercial entity.

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.

11. Acknowledgements

Special thanks to Bernie Hoeneisen, Enrico Tomae, Stephen Farrel, Brian Trammell and Neal Walfield for roughly reviewing first versions of this Internet-Draft and providing valuable feedback and patches.

[[Much more general acknowledgements to follow.]]

12. References

12.1. 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>.

[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>.

[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>.

[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>.

[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>.

12.2. Informative References

[EnigmailpEpSrc] Enigmail project, "Source code for Enigmail/pEp", June 2017, <https://enigmail.net/index.php/en/download/source-code>.

[pEpCore] pEp Foundation, "Core source code and reference implementation of pEp (engine and adapters)", June 2017, <https://letsencrypt.pep.foundation/dev/>.

[pEpEmail] pEp Foundation, "pEp email [Early Internet-Draft]", June 2017, <https://letsencrypt.pep.foundation/trac/browser/internet-drafts/pep-email/draft-birk-pep-email-NN.txt>.

[pEpForAndroidSrc] pEp Security, "Source code for pEp for Android", June 2017, <https://cacert.pep-security.lu/gitlab/android/pep>.

[pEpForiOSSrc] pEp Security, "Source code for pEp for iOS", June 2017, <https://cacert.pep-security.ch/dev/repos/pEp_for_iOS/>.

[pEpForOutlookSrc] pEp Security, "Source code for pEp for Outlook", June 2017, <https://cacert.pep-security.lu/dev/repos/pEp_for_Outlook/>.

[pEpKeySync]
pEp Foundation, "pEp Key Synchronization Protocol [Early Internet-Draft]", June 2017, <https://letsencrypt.pep.foundation/trac/browser/internet-drafts/pep-keysync/draft-birk-pep-keysync-NN.txt>.

[pEpTrustRating]
pEp Foundation, "pretty Easy privacy (pEp): Trust Rating System [Early Internet-Draft]", June 2017, <https://letsencrypt.pep.foundation/trac/browser/internet-drafts/pep-rating/draft-birk-pep-rating-NN.txt>.

[pEpTrustwords]
pEp Foundation, "pretty Easy privacy (pEp): Trustwords concept [Early Internet-Draft]", June 2017, <https://letsencrypt.pep.foundation/trac/browser/internet-drafts/pep-trustwords/draft-birk-pep-trustwords-NN.txt>.

Authors' Addresses

Volker Birk
pEp Foundation

Email: vb@pep-project.org

Hernani Marques
pEp Foundation

Email: hernani.marques@pep.foundation

Shelburn
pEp Foundation

Email: shelburn@pep.foundation

Sandro Koechli
pEp Security

Email: sandro@pep-security.net