

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 29, 2018

V. Birk
H. Marques
S. Shelburn
pEp Foundation
June 27, 2018

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

Abstract

Building on already available security formats and message transports (like PGP/MIME for email), and with the intention to stay interoperable to systems widely 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 relies on "Trustwords" (as a word mapping 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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 29, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terms	4
3. Protocol's Core Design Principles	4
3.1. Privacy by Default	4
3.2. Interoperability	5
3.3. Peer-to-Peer (P2P)	5
3.4. User Experience (UX)	6
4. pEp identity system	6
4.1. Terms	7
4.1.1. Key	7
4.1.2. User	7
4.1.3. Address	7
4.1.4. Identity	7
4.2. Example: Difference between pEp and OpenPGP	8
5. Key Management	8
5.1. Private Keys	9
5.2. Public Key Distribution	9
5.3. Passphrases	10
6. Privacy Status	10
7. Options in pEp	11
7.1. Option "Passive Mode"	11
7.2. Option "Disable Protection"	11
7.2.1. For all communications	11
7.2.2. For some communications	11
7.3. Option "Extra Keys"	12
7.4. Option "Blacklist Keys"	12
7.5. Establishing trust between peers	12
8. Security Considerations	12
9. Implementation Status	13
9.1. Introduction	13
9.2. Reference implementation of pEp's core	13
9.3. Abstract Crypto API examples	14
9.4. Current software implementing pEp	15
10. Notes	15
11. Acknowledgments	16
12. References	16

12.1. Normative References	16
12.2. Informative References	16
Appendix A. Excerpts from the pEp Reference Implementation . . .	18
A.1. pEp Identity	18
A.1.1. Corresponding SQL	19
A.2. pEp Communication Type	20
A.3. Abstract Crypto API examples	21
A.3.1. Encrypting a Message	21
A.3.2. Decrypting a Message	22
A.3.3. Obtain Common Trustwords	24
Appendix B. Document Changelog	24
Appendix C. Open Issues	25
Authors' Addresses	25

1. Introduction

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 communications without depending on centralized infrastructures.

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

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

[[Note: 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 particular 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. Therefore, data minimization is a primary goal in

pEp (e.g., omitting unnecessary email headers and encrypting the subject line).

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 the different standardized and/or widely spread communications channels, such that users can reach their peers in the most privacy-enhancing way possible.

[[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.]]

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].

- o Handshake: The process when Alice - e.g. in-person or via phone - contacts Bob to verify Trustwords (or by fallback: fingerprints) is called handshake. [E-D.birk-pep-handshake]
- o Trustwords: A scalar-to-word representation of 16-bit numbers (0 to 65535) to natural language words. When doing a handshake, peers are shown combined Trustwords of both public keys involved to ease the comparison. [I-D.birk-pep-trustwords]
- o Trust on First Use (TOFU): cf. [RFC7435]
- o Man-in-the-middle attack (MITM): cf. [RFC4949]

3. Protocol's Core Design Principles

3.1. Privacy by Default

The pEp protocols are about to ensure privacy. However, there are cases where privacy and security are contradicting, e.g., in PGP's Web of Trust, because relations between people and trust levels are leaked. Also query privacy is not ensured in such a model when obtaining keys from remote locations. Whenever security and privacy fit together, highest security and privacy is to be reached. However, where they contradict each other, privacy is always to be

chosen over security. Though, users SHOULD have the choice to override the default by corresponding options.

In pEp messaging (e.g., when using HTML) content SHALL NOT be obtained from remote locations as this constitutes a privacy breach.

3.2. Interoperability

The pEp propositions seek to be interoperable with message formats and cryptography already widespread. Seamless communication between users of software, which implements pEp and other messaging tools for end-to-end encryption, is a design goal.

Therefore:

- o Be conservative (strict) in requirements for pEp implementations and how they behave between each other.
- o Be liberal (accepting) in what comes in from non-pEp implementations (e.g., do not send, but support to decipher PGP/INLINE formats).
- o 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).

3.3. Peer-to-Peer (P2P)

Messaging and verification processes in pEp are designed to work Peer-to-Peer (P2P) without intermediaries in between.

This means, there MUST 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 (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) SHALL NOT be signaled as trustworthy (cf. [E-D.birk-pep-trust-rating]) to users of pEp (e.g., when interoperating with peers using S/MIME) by default.

3.4. User Experience (UX)

Implementers of pEp MUST take special care not to confuse users with technical terms, especially those of cryptography (e.g., "keys", "certificates" or "fingerprints"), unless users explicitly ask for such terms; i.e., advanced settings MAY be available, in some cases further options may even be required. However, those SHALL NOT be unnecessarily exposed to users of pEp implementations at the first sight.

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, i.e., a goal of pEp is that users 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 ensure 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 ensure a Privacy Status is clearly visible to the user - on contact as well as on message level - so that users easily understand, which level of privacy messages are about to be sent or were received with, respectively.

[[Note: 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.]]

4. pEp identity system

In pEp, users MUST have the possibility to have different identities.

pEp users MUST have the option to choose different identities. This allows an Internet user to decide how to reveal oneself to the world and is an important element to achieve privacy.

The different identities MUST NOT correlate with other by default. On the other hand, combining different identities MUST be supported (to support aliases).

4.1. Terms

4.1.1. Key

A key is an OpenPGP compatible asymmetric key pair. Other formats and temporary symmetrical keys can be generated by Key Mapping.

Keys in pEp are identified by the full fingerprint (fpr) of its public key.

4.1.2. User

A 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 user is identified by a user ID (user_id). The user_id SHOULD be an UUID, it MAY be an arbitrary unique string.

The own user can have a user_id like all other users. If it doesn't, then it has PEP_OWN_USERID "pEp_own_userId" as user_id.

A user can have a default key.

4.1.3. Address

An address is a network address, e.g., an SMTP address or another URI.

[[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.]]

4.1.4. Identity

An identity is a (possibly pseudonymous) representation of a user encapsulating how this user appears in the network.

An identity is defined by the mapping of user_id to address. If no user_id is known, it is guessed by mapping of username and address.

An identity can have a temporary user_id as a placeholder until a real user_id is known.

An identity can have a default key.

[[Note: 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 Appendix A.1 you can find how a pEp identity is defined in the reference implementation of the pEp Engine.

4.2. Example: Difference 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

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

A pEp implementation MUST ensure cryptographic keys for end-to-end cryptography are generated for every identity configured (or instantly upon its configuration [[TODO: unclear/rewrite/simplify]]), 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. [[TODO: rewrite/simplify]]

pEp includes a Trust Rating system [E-D.birk-pep-trust-rating] defining what kind of encryption is 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 insecure (e.g, insufficient key length), pEp implementers are REQUIRED to generate new keys for use with the respective identity.

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.

5.1. Private Keys

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 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.

[[Note: 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.]]

A decentralized proposition - the pEp Key Synchronization protocol. [E-D.birk-pep-keysenc] - defines how pEp users can distribute their private keys among different devices 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. Public Key Distribution

Implementers of pEp are REQUIRED to ensure that the identity's public key is attached to every outgoing message. However, this MAY be omitted if the peer previously received a message encrypted with the public key of the sender.

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.

Implementers of pEp SHALL prevent that public keys attached to messages (e.g, in email) are displayed to the user, in order to avoid users getting confused by a file they cannot potentially deal with.

Metadata (e.g., email headers) MUST NOT be added to announce a user's public key. This is considered unnecessary information leakage as it may affect users' privacy, which depends also on a country's data retention laws. Furthermore, this affects interoperability to existing users (e.g., in the OpenPGP field) that 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 of received messages (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 provided with 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 pEp clients SHALL NOT be uploaded to any (intermediate) keystore locations without the user's explicit consent.

5.3. Passphrases

Passphrases to protect a 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) 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 SHALL 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.

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

- o a communication channel with a given peer was or ought to be and

- o 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 representations to be followed is outlined in [E-D.birk-peg-trust-rating].

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 5.2). 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 available.

7.2. Option "Disable Protection"

This option SHALL not affect the user's ability to decipher already received or sent messages. [[TODO: Public key added in these cases?]]

Protection can be disabled generally or selectively.

7.2.1. For all communications

Implementers of pEp MUST provide an option "Disable Protection" for the user's choice to disable any outgoing encryption and signing.

7.2.2. For some communications

Implementers of pEp MUST provide an option to allow users to disable protection (encryption and signing) for specific contacts or messages.

7.3. Option "Extra Keys"

For internal environments there may be a need to centrally decrypt persons' messages for archiving or other legal purposes (e.g., in the contexts of public offices and enterprises) by authorized personnel. Therefore, pEp implementers MAY provide an "Extra Keys" option where a message gets encrypted with at least one additional public key. The corresponding (shared) secret(s) to decrypt are intended to be held - safely - by CISO staff and/or other authorized personnel for such an organization. [[TODO: Shared secret? no private key?]]

The Extra Keys feature MUST NOT be activated by default for any network address and is intended to be an option only for organizational identities and their corresponding network addresses and accounts - not for addresses used for private purposes. That is, the Extra Keys feature is a feature which SHOULD NOT apply to all identities a user might possess, even if activated.

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 [I-D.birk-pep-trustwords] are used for users to compare the authenticity of peers in order to mitigate 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. 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 an 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.

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. [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:

- o 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.

- o 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.

- o 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.

- o 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:

- o pEp COM Server Adapter
- o pEp JNI Adapter
- o pEp JSON Adapter
- o pEp ObjC (and Swift) Adapter
- o pEp Python Adapter
- o pEp Qt Adapter

9.3. 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.

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 described nearer in [E-D.birk-pep-email]:

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

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:

- o Software using the trademarks MUST be backdoor-free.
- o 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. Acknowledgments

The authors would like to thank the following people who have provided feedback or significant contributions to the development of this document: Bernie Hoeneisen, Brian Trammell, Enrico Tomae, Eric Rescorla Neal Walfield, and Stephen Farrel.

[[TODO: Add those who commented on mailing list (on this draft) as well as those who provided feedback in person or during BarBoFs.]]

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]

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>>.
- [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>>.

12.2. Informative References

- [E-D.birk-pep-email] Birk, V. and H. Marques, "pretty Easy privacy (pEp): Secure and Trusted Email Communication", June 2018, <<https://pep.foundation/dev/repos/internet-drafts/file/tip/pep-email/draft-birk-pep-email-NN.txt>>.

Early draft

- [E-D.birk-pep-handshake] Marques, H., "pretty Easy privacy (pEp): Contact Authentication through Handshake", June 2018, <<https://pep.foundation/dev/repos/internet-drafts/file/tip/pep-handshake/draft-marques-pep-handshake-00.txt>>.

Early draft

[E-D.birk-pep-keysenc]

Birk, V. and H. Marques, "pretty Easy privacy (pEp): Key Synchronization Protocol", June 2018, <<https://pep.foundation/dev/repos/internet-drafts/file/tip/pep-keysenc/draft-birk-pep-keysenc-NN.txt>>.

Early draft

[E-D.birk-pep-trust-rating]

Birk, V. and H. Marques, "pretty Easy privacy (pEp): Trust Rating System", June 2018, <<https://pep.foundation/trac/browser/internet-drafts/pep-rating/draft-marques-pep-rating-00.txt>>.

Early draft

[I-D.birk-pep-trustwords]

Birk, V., Marques, H., and B. Hoeneisen, "IANA Registration of Trustword Lists: Guide, Template and IANA Considerations", draft-birk-pep-trustwords-02 (work in progress), June 2018.

[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/>>.

[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>>.

[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>>.

```
[SRC.enigmailpep]
    "Source code for Enigmail/pEp", June 2018,
    <https://enigmail.net/index.php/en/download/source-code>.

[SRC.pepcore]
    "Core source code and reference implementation of pEp
    (engine and adapters)", June 2018,
    <https://pep.foundation/dev/>.

[SRC.pepforandroid]
    "Source code for pEp for Android", June 2018,
    <https://pep-security.lu/gitlab/android/pep>.

[SRC.pepforios]
    "Source code for pEp for iOS", June 2018,
    <https://pep-security.ch/dev/repos/pEp_for_iOS/>.

[SRC.pepforoutlook]
    "Source code for pEp for Outlook", June 2018,
    <https://pep-security.lu/dev/repos/pEp_for_Outlook/>.
```

Appendix A. Excerpts from the pEp Reference Implementation

This section provides excerpts of the running code from the pEp reference implementation pEp engine (C99 programming language). [[TODO: Maybe rewrite sentence a bit]]

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
    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.1.1.1. 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,  
    device_group 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,  
    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  
);  
CREATE INDEX pgp_keypair_expires on pgp_keypair (  
    expires  
);
```

A.2. pEp Communication Type

In the following, an example for the rating of communication types, the definition of the 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_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 compromised 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;
```

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() - 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 if no
//                       encryption could take place
//     enc_format (in)   encrypted format
//     flags (in)        flags to set special encryption features
//
// return value:
//     PEP_STATUS_OK      on success
//     PEP_KEY_HAS_AMBIG_NAME at least one of the recipient
//                       keys has an ambiguous name
//     PEP_UNENCRYPTED     no recipients with usable key,
//                       message is left unencrypted,
//                       and key is attached to it
//
// caveat:
//     the ownership 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
);

```

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

A.3.2. Decrypting a Message

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

```

// 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_CANNOT_REENCRYPT if message was decrypted (and possibly
//     verified) but a reencryption operation is expected by the
//     caller and failed
//     or PEP_STATUS_OK on success
//
// flag values:
//     in:
//         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
//     out:
//         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 src object has been modified. At the
//             moment, this is always as a direct result of the
//             behaviour driven by the input flags. This flag is the
//             ONLY value that should be relied upon to see if such
//             changes have taken place.
//         PEP_decrypt_flag_consume
//             used by sync
//         PEP_decrypt_flag_ignore
//             used by sync
//
// caveat:
//     the ownership of 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.
//     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
```

```
);
```

A.3.3. Obtain Common Trustwords

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

```
// 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
);
```

Appendix B. Document Changelog

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

o 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
- o draft-birk-peg-01:
 - * Mostly editorial
 - o draft-birk-peg-00:
 - * Initial version

Appendix C. Open Issues

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

- o Better explain Passive Mode
- o Better explain / illustrate pEp's identity system
- o Explain Key Mapping

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/>

Shelburn
pEp Foundation
Oberer Graben 4
CH-8400 Winterthur
Switzerland

Email: shelburn@pep.foundation
URI: <https://pep.foundation/>

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 28, 2018

V. Birk
H. Marques
pEp Foundation
B. Hoeneisen
Ucom.ch
June 26, 2018

IANA Registration of Trustword Lists: Guide, Template and IANA
Considerations
draft-birk-pep-trustwords-02

Abstract

This document specifies the IANA Registration Guidelines for Trustwords, describes corresponding registration procedures, and provides a guideline for creating Trustword list specifications.

Trustwords are common words in a natural language (e.g., English) to which the hexadecimal strings are mapped to. This makes verification processes (e.g., comparison of fingerprints), more practical and less prone to misunderstandings.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 28, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terms	3
3. The Concept of Trustword Mapping	3
3.1. Example	3
3.2. Previous work	4
3.3. Number of Trustwords for a language	4
3.4. Language	5
3.5. The nature of the words	5
4. IANA Considerations	5
4.1. Registration Template (XML chunk)	6
4.2. IANA Registration	7
4.2.1. Language Code (<languagecode>)	7
4.2.2. Bit Size (<bitsize>)	7
4.2.3. Number Of Unique Words (<numberofuniquewords>)	7
4.2.4. Bijectivity (<bijective>)	8
4.2.5. Version (<version>)	8
4.2.6. Registration Document(s) (<registrationdocs>)	8
4.2.7. Requesters (<requesters>)	8
4.2.8. Further Information (<additionalinfo>)	9
4.2.9. Wordlist (<wordlist>)	9
5. Security Considerations	10
6. Acknowledgements	10
7. References	10
7.1. Normative References	10
7.2. Informative References	10
Appendix A. IANA XML Template Example	12
Appendix B. Document Changelog	13
Appendix C. Open Issues	13
Authors' Addresses	14

1. Introduction

In public-key cryptography comparing the public keys' fingerprints of the communication partners involved is vital to ensure that there is no man-in-the-middle (MITM) attack on the communication channel. Fingerprints normally consist of a chain of hexadecimal chars. However, comparing hexadecimal strings is often impractical for regular human users and prone to misunderstandings.

To mitigate these challenges, several systems offer the comparison of Trustwords as an alternative to hexadecimal strings. Trustwords are common words in a natural language (e.g., English) to which the hexadecimal strings are mapped to. This makes the verification process more natural for human users.

For example, in pEp's proposition of Privacy by Default [I-D.birk-pep] Trustwords are used to achieve easy contact verification for end-to-end encryption. Trustword comparison is offered after the peers have exchanged public keys opportunistically. Examples for Trustword lists used by current pEp implementations can be found in CSV format, here:
<https://pep.foundation/dev/repos/pEpEngine/file/tip/db>.

In addition to contact verification, Trustwords are also used for other purposes, such as Human-Readable 128-bit Keys [RFC1751], One Time Passwords (OTP) [RFC1760] [RFC2289], SSH host-key verification, VPN Server certificate verification, and to import or synchronize secret key across different devices of the same user [E-D.birk-pep-keysync]. Further ideas include to use Trustwords for contact verification in Extensible Messaging and Presence Protocol (XMPP) [RFC6120], for X.509 [RFC3647] certificate verification in browsers or in block chain applications for crypto currencies.

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].

- o Handshake: The process when Alice - e.g. in-person or via phone - contacts Bob to verify Trustwords (or by fallback: fingerprints) is called handshake. [E-D.birk-pep-handshake]
- o Man-in-the-middle attack (MITM): cf. [RFC4949]

3. The Concept of Trustword Mapping

3.1. Example

A fingerprint typically looks like:

F482 E952 2F48 618B 01BC 31DC 5428 D7FA ACDC 3F13

Its mapping to English Trustwords could look like:

dog house brother town fat bath school banana kite task

Or its mapping to German Trustwords could like like:

klima gelb lappen weg trinken alles kaputt rasen rucksack durch

Instead of the former hexadecimal string, users can compare ten common words of their language.

Note: This examples are for illustration purposes only and do not make use any any published Trustword list.

3.2. Previous work

The basic concept of Trustwords mapping has been already documented in the past, e.g. for use in One-Time Passwords (OTP) [RFC1751] [RFC1760] [RFC2289] or the PGP Word List ("Pretty Good Privacy word list" [PGP.wl], also called a biometric word list, to compare fingerprints.

Regarding today's needs, previous proposals have the following shortcomings:

- o Limited number of Trustwords (small Trustword dictionaries), which generally results in more Trustwords to be compared
- o Usually only available in English language, which does not normally allow its usage by non-English speakers in a secure manner

Furthermore, there are differences in the basic concept:

- o This work allows for better tailoring the target audience to ordinary human users, i.e. not technical stuff (or IT geeks) only.
- o As in many usage scenarios the Trustwords are only read (and compared), but not written down nor typed in by humans, there is a less strong need to keep the Trustwords themselves short. One such scenario is to use a side channel (e.g. phone) to compare the Trustwords. In fact longer Trustwords increases increase the entropy, as the dictionary is larger and the likelihood for phonetic collision can be decreased.

3.3. Number of Trustwords for a language

If the number of Trustwords is low, a lot of Trustwords need to be compared, which make a comparison somewhat cumbersome for users. This may lead to degraded usability.

To reduce the number of Trustwords to compare, in pEp's proposition of Privacy by Default [I-D.birk-pep] 16-bit scalars are mapped to natural language words. Therefore, the size (by number of key - value pairs) of any key - value pair structure is 65536. However, the number of unique values to be used in a language may be less than 65536. This can be addressed e.g. by using the same value (Trustword) for more than one key. In these cases, the entropy of the representation is slightly reduced. (Example: A Trustwords list of just 42000 words still allows for an entropy of $\log_2(42000) \approx 15.36$ bits in 16-bit mappings.)

On the other hand, small sized Trustword lists allow for Trustwords with shorter strings, which are easier to use in scenarios where Trustwords have to be typed in e.g. OTP applications.

The specification allows for different dictionary sizes.

3.4. Language

Although English is rather widespread around the world, the vast majority of the world's population does not speak English. For an application to be useful for ordinary people, localization is a must. Thus, Trustword lists in different languages can be registered.

For applications where two humans establish communication it is very likely that they share a common language. So far no real use case for translations between Trustword lists in different languages has been identified. As translations also drastically increase the complexity for IANA registrations, translations of Trustwords beyond the scope of this document.

3.5. The nature of the words

Every Trustwords list SHOULD be cleared from swearwords in order to not offend users. This is a task to be carried out by speakers of the respective natural language (i.e., by native language speakers).

4. IANA Considerations

Each natural language requires a different set of Trustwords. To allow implementers for identical Trustword lists, a IANA registry is to be established. The IANA registration policy according to [RFC8126] is "Expert Review" and "Specification Required".

[[Note: Further details of the IANA registry and requirements for the expert to assess the specification are for further study. A similar approach as used in [RFC6117] is likely followed.]]

4.1. Registration Template (XML chunk)

```
<record>
  <languagecode>
    <!-- ISO 639-3 (e.g. eng, deu, ...) -->
  </languagecode>
  <bitsize>
    <!-- How many bits can be mapped with this list
         (e.g. 8, 16, ...) -->
  </bitsize>
  <numberofuniquewords>
    <!-- number of unique words registered
         (e.g. 256, 65536, ...) -->
  </numberofuniquewords>
  <bijective>
    <!-- whether or not the list allows for a two-way-mapping
         (e.g. yes, no) -->
  </bijective>
  <version>
    <!-- version number within language
         (e.g. b.1.2, n.0.1, ...) -->
  </version>
  <registrationdocs>
    <!-- Change accordingly -->
    <xref type="rfc" data="rfc2551"/>
  </registrationdocs>
  <requesters>
    <!-- Change accordingly -->
    <xref type="person" data="John_Doe"/>
    <xref type="person" data="Jane_Dale"/>
  </requesters>
  <additionalinfo>
    <paragraph>
      <!-- Text with additional information about
           the Wordlist to be registered -->
    </paragraph>
    <artwork>
      <!-- There can be artwork sections, too -->
    </artwork>
  </additionalinfo>
  <wordlist>
    <!-- Change accordingly -->
    <w0>first</w0>
    <w1>second</w1>
    [...]
    <w65535>last</w65535>
  </wordlist>
</record>
```



```
<people>
  <person id="John_Doe">
    <name> <!-- Firstname Lastname --> </name>
    <org> <!-- Organization Name --> </org>
    <uri> <!-- mailto: or http: URI --> </uri>
    <updated> <!-- date format YYYY-MM-DD --> </updated>
  </person>
  <!-- repeat person section for each person -->
</people>
```

Authors of a Wordlist are encouraged to use these XML chunks as a template to create the IANA Registration Template.

4.2. IANA Registration

An IANA registration will contain the following elements:

4.2.1. Language Code (<languagecode>)

The language code follows the ISO 639-3 specification [ISO693], e.g., eng, deu.

[[Note: It is for further study, which of the ISO 639 Specifications is most suitable to address the Trustwords' challenge.]]

Example usage for German:

e.g. <languagecode>deu</languagecode>

4.2.2. Bit Size (<bitsize>)

The bit size is the number of bits that can be mapped with the Wordlist. The number of registered words in a word list MUST be $2^{\text{(<bitsize>)}}$.

Example usage for 16-bit Wordlist:

e.g. <bitsize>16</bitsize>

4.2.3. Number Of Unique Words (<numberofuniquewords>)

The number of unique words that are registered.

e.g. <numberofuniquewords>65536</numberofuniquewords>

4.2.4. Bijectivity (<bijective>)

Whether the registered Wordlist has a one-to-one mapping, meaning the number of unique words registered equals $2^{<bitsize>}$.

Valid content: (yes | no)

e.g. <bijective>yes</bijective>

4.2.5. Version (<version>)

The version of the Wordlist MUST be unique within a language code.

[[Note: Requirements to a "smart" composition of the version number are for further study]]

e.g. <version>b.1.2</version>

4.2.6. Registration Document(s) (<registrationdocs>)

Reference(s) to the Document(s) containing the Wordlist

e.g. <registrationdocs>
 <xref type="rfc" data="rfc4979"/>
</registrationdocs>

e.g. <registrationdocs>
 <xref type="rfc" data="rfc8888"/> (obsoleted by RFC 9999)
 <xref type="rfc" data="rfc9999"/>
</registrationdocs>

e.g. <registrationdocs>
 [International Telecommunications Union,
 "Wordlist for Foofoo application",
 ITU-F Recommendation B.193, Release 73, Mar 2009.]
</registrationdocs>

4.2.7. Requesters (<requesters>)

The persons requesting the registration of the Wordlist. Usually these are the authors of the Wordlist.

```
e.g. <requesters>
      <xref type="person" data="John_Doe"/>
    </requesters>

    <people>
      <person id="John_Doe">
        <name>John Doe</name>
        <org>Example Inc.</org>
        <uri>mailto:john.doe@example.com</uri>
        <updated>2018-06-20</updated>
      </person>
    </people>
```

Note: If there is more than one requester, there must be one <xref> element per requester in the <requesters> element, and one <person> chunk per requester in the <people> element.

4.2.8. Further Information (<additionalinfo>)

Any other information the authors deem interesting.

```
e.g. <additionalinfo>
      <paragraph>more info goes here</paragraph>
    </additionalinfo>
```

Note: If there is no such additional information, then the <additionalinfo> element is omitted.

4.2.9. Wordlist (<wordlist>)

The full Wordlist to be registered. The number of words MUST be a power of 2 as specified above. The element names serve as key used for enumeration of the Trustwords (starting at 0) and the elements contains the values being individual natural language words in the respective language.

```
e.g. <wordlist>
      <w0>first</w0>
      <w1>second</w1>
      [...]
      <w65535>last</w65535>
    </wordlist>
```

```
] ]>
```

[[Note: The exact representation of the Wordlist is for further study.]]

5. Security Considerations

There are no special security considerations.

6. Acknowledgements

The authors would like to thank the following people who have provided feedback or significant contributions to the development of this document: Andrew Sullivan, Claudio Luck, Daniel Kahn Gilmore, Michael Richardson, Rich Salz, and Yoav Nir.

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]

7. References

7.1. 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>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.

7.2. Informative References

- [E-D.birk-pep-handshake] Marques, H., "pretty Easy privacy (pEp): Contact Authentication through Handshake", June 2018, <<https://pep.foundation/dev/repos/internet-drafts/file/tip/pep-handshake/draft-marques-pep-handshake-00.txt>>.

Early draft

[E-D.birk-pep-keysenc]

Birk, V. and H. Marques, "pretty Easy privacy (pEp): Key Synchronization Protocol", June 2018, <<https://pep.foundation/dev/repos/internet-drafts/file/tip/pep-keysenc/draft-birk-pep-keysenc-NN.txt>>.

Early draft

[I-D.birk-pep]

Birk, V., Marques, H., Shelburn, S., and S. Koechli, "pretty Easy privacy (pEp): Privacy by Default", draft-birk-pep-01 (work in progress), January 2018.

[ISO639]

"Language codes - ISO 639", n.d., <<https://www.iso.org/iso-639-language-codes.html>>.

[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/>>.

[PGP.wl]

"PGP word list", November 2017, <https://en.wikipedia.org/w/index.php?title=PGP_word_list&oldid=749481933>.

[RFC1751]

McDonald, D., "A Convention for Human-Readable 128-bit Keys", RFC 1751, DOI 10.17487/RFC1751, December 1994, <<https://www.rfc-editor.org/info/rfc1751>>.

[RFC1760]

Haller, N., "The S/KEY One-Time Password System", RFC 1760, DOI 10.17487/RFC1760, February 1995, <<https://www.rfc-editor.org/info/rfc1760>>.

[RFC2289]

Haller, N., Metz, C., Nesser, P., and M. Straw, "A One-Time Password System", STD 61, RFC 2289, DOI 10.17487/RFC2289, February 1998, <<https://www.rfc-editor.org/info/rfc2289>>.

[RFC3647]

Chokhani, S., Ford, W., Sabett, R., Merrill, C., and S. Wu, "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework", RFC 3647, DOI 10.17487/RFC3647, November 2003, <<https://www.rfc-editor.org/info/rfc3647>>.

- [RFC6117] Hoeneisen, B., Mayrhofer, A., and J. Livingood, "IANA Registration of Enumservices: Guide, Template, and IANA Considerations", RFC 6117, DOI 10.17487/RFC6117, March 2011, <<https://www.rfc-editor.org/info/rfc6117>>.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <<https://www.rfc-editor.org/info/rfc6120>>.

Appendix A. IANA XML Template Example

This section contains a non-normative example of the IANA Registration Template XML chunk.

```
<record>
  <languagecode>lat</languagecode>
  <bitsize>16</bitsize>
  <numberofuniquewords>57337</numberofuniquewords>
  <bijective>no</bijective>
  <version>n.0.1</version>
  <registrationdocs>
    <xref type="rfc" data="rfc2551"/>
  </registrationdocs>
  <requesters>
    <xref type="person" data="Julius_Caesar"/>
  </requesters>
  <additionalinfo>
    <paragraph>
      This Wordlist has been optimized for
      the Roman Standards Process.
    </paragraph>
  </additionalinfo>
  <wordlist>
    <w0>errare</w0>
    <w1>humanum</w1>
    [...]
    <w65535>est</w65535>
  </wordlist>
</record>

<people>
  <person id="Julius_Caesar">
    <name>Julius Caesar</name>
    <org>Curia Romana</org>
    <uri>mailto:julius.cesar@example.com</uri>
    <updated>1999-12-31</updated>
  </person>
</people>
```

Appendix B. Document Changelog

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

- o draft-birk-pep-trustwords-02:
 - * Minor editorial changes and bug fixes
 - * Added more items to Open Issues
 - * Add usage example
- o draft-birk-pep-trustwords-01:
 - * Included feedback from mailing list and IETF-101 SECDISPATCH WG, e.g.
 - + Added more explanatory text / less focused on the main use case
 - + Bit size as parameter
 - * Explicitly stated translations are out-of-scope for this document
 - * Added draft IANA XML Registration template, considerations, explanation and examples
 - * Added Changelog to Appendix
 - * Added Open Issue section to Appendix

Appendix C. Open Issues

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

- o More explanatory text for Trustword use cases, properties and requirements
- o Further details of the IANA registry and requirements for the expert to assess the specification
- o Decide which ISO language code either 639-1 or 639-3 to use, i.e., ISO-639-1 (e.g., ca, de, en, ...) as currently used in pEp implementations (running code) or ISO-639-3 (eng, deu, ita, ...)
- o Adjust exact representation of wordlists

- * e.g. XML, CSV, ...
- * Syntax for non-ASCII letters or language symbols (UTF-8) in Wordlists
- o Need for optional entropy value assigned to words, to account for similar phonetics among words in the same wordlist?
- o Need for an additional field, to define what a wordlist is optimized for, e.g., "entropy", "minimize word lengths", ...?
- o Work out (requirements for) "smart" composition of the version number
- o Decide whether in non-bijective Wordlists the redundant words need to be repeated in the IANA Registration
- o Register only a hash over the wordlist with IANA?
- o Does it make sense to open registrations for other patterns than just words, e.g., images?

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
Ucom Standards Track Solutions GmbH
CH-8046 Zuerich
Switzerland

Phone: +41 44 500 52 44

Email: bernie@ietf.hoeneisen.ch (bernhard.hoeneisen AT ucom.ch)

URI: <https://ucom.ch/>

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 October 2022

P. M. Hallam-Baker
20 April 2022

DNS Web Service Discovery
draft-hallambaker-web-service-discovery-07

Abstract

This document describes a standardized approach to discovering Web Service Endpoints from a DNS name. Services are advertised using the DNS SRV and TXT records and the HTTP Well Known Service conventions.

This document is also available online at
<http://mathmesh.com/Documents/draft-hallambaker-web-service-discovery.html>.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 22 October 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction	2
2. Definitions	3
2.1. Requirements Language	3
2.2. Defined Terms	3
3. Service Discovery	4
3.1. Host Identification	4
3.1.1. SRV Host discovery	4
3.2. Service Description	4
3.2.1. TXT Service and Host Description	4
3.3. Service Selection	5
3.4. Web Service Endpoint Determination	5
3.5. DNS Fallback	5
3.6. Example	6
4. Further Work	7
4.1. Additional Description Keys	7
4.2. Service Scaling	8
5. Security Considerations	8
6. IANA Considerations	8
6.1. Well-Known URIs	8
7. Normative References	9
8. Informative References	9

1. Introduction

Web services are traditionally identified by means of a URI specifying a Web Service Endpoint (WSE). This approach is unsatisfactory in many situations:

- * Specification of the Web Service requires the transport and presentation protocols to be fixed.
- * The discovery mechanism does not provide support for load balancing or fault tolerance.
- * The identifiers are unsuited for human interaction.

The last consideration is a particular concern where an account identifier is exposed to the user. Attempts to 'teach' users to use URIs as account identifiers have been predictably unsuccessful. Users expect and require accounts to be of the form `user@example.com` and not `http://service.example.com/service/user`.

The Web Service discovery process described in this specification builds on the approach specified in DNS-Based Service Discovery [RFC6763]. This uses DNS SRV records as the basis for service discovery and TXT records as the basis for service description. This

approach allows Web Services to make use of the load balancing and fault tolerance features of SRV and the service negotiation capabilities provided by the service description.

One difficulty that is frequently encountered in attempting to make use of DNS records for service discovery is that it is not always possible for an application process to access this information. Specifications address the world as it actually is rather than as some believe it should be have proven more robust in real world deployment than those that do not. The discovery process defined includes a fallback strategy to enable clients to achieve Web Service discovery in these circumstances.

Another difficulty that is encountered is that the SRV record maps service names to host names rather than Web Service Endpoints. A convention is thus required to map a host name and protocol prefix to a Web Service Endpoint. The HTTP Well Known Service [RFC5785] mechanism is used for this purpose.

While the approach adopted in this specification closely follows that of [RFC6763], there is an important difference in that the earlier specification sets out a framework which Web Services may apply to develop a discovery approach that suits their particular needs while this specification defines exactly one such approach. In particular, the use of a common set of TXT keys to specify service parameters enables service discovery and negotiation to be delegated to common support libraries rather than being implemented independently in each application.

2. Definitions

This section presents the related specifications and standard, the terms that are used as terms of art within the documents and the terms used as requirements language.

2.1. Requirements Language

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

2.2. Defined Terms

Web Service An Internet service provided by one or more Web Service Hosts that are addressable by a single Web Service Endpoint and are intended to provide logically equivalent services.

Web Service Endpoint (WSE) A URI that specifies a Web Service or Web

Service Host.

Web Service Host The actual machine (physical or virtual) that provides a Web Service

3. Service Discovery

Service discovery is the process of resolving the address of a Web Service to a Web Service Endpoint, a URI [RFC3986] at which the service is provided.

3.1. Host Identification

The first step in service discovery is to resolve the <domain> and <service> identifiers to the IP address of a host that provides that service.

3.1.1. SRV Host discovery

A client attempting to connect to the service first attempts to locate an SRV record [RFC2782] for the specified service:

```
_<service>._tcp.<domain> SRV <priority> <weight> <port> <host>
```

Where <service> is the IANA assigned service name, <priority> and <weight> are the SRV priority and weight parameters specified in [RFC2782], <port> is the TCP port number and <host> is the DNS name of the host for which the service advertisement is made.

If no SRV records are found, the client MAY abort the connection or attempt use of the Fallback Discovery process described below.

3.2. Service Description

The second step in service discovery is to identify the attributes of the Web Service and Web Service Hosts providing that service.

3.2.1. TXT Service and Host Description

A service MAY advertise service and/or host description information using TXT records as described in DNS-Based Service Discovery [RFC6763] . These have the following format:

```
_<service>._tcp.<domain> TXT "<key>=<value> [<tag>=<value>]*"  
_<service>._tcp.<host>  TXT "<key>=<value> [<tag>=<value>]*"
```

Where <domain> and <host> are the domain names specified in the corresponding SRV records.

Service descriptions specified under the domain address of the service apply to all host instances of the service. Descriptions specified under the domain address of a host instance apply only to that host instance and take precedence over values specified at the service level.

The following keys are currently defined:

`path` The path to use to construct the Web Service Endpoint.

`version` The service version(s) supported in the format `<max>-<min>`

`encoding` An IANA media type specifying a supported encoding format

3.3. Service Selection

Web Service Hosts that do not meet the requirements of the client attempting to create a connection are eliminated before applying SRV service selection criteria specified in [RFC2782].

Clients SHOULD limit the number of connections attempted before abandoning the attempt to connect.

3.4. Web Service Endpoint Determination

Having selected a Web Service Host, the client determines the Web Service Endpoint as follows:

- * If the description of the host specifies a `path` key, the corresponding value is used as the path, otherwise,
- * if the description of the service specifies a `path` key, the corresponding value is used as the path, otherwise,
- * the path is `/.well-known/srv/<service>`

3.5. DNS Fallback

Despite the fact that SRV records have been a part of the DNS standard for 20 years, it is not uncommon for network intermediaries to implement SRV record resolution incorrectly or block it entirely. If no SRV record is found, a client MAY perform fallback discovery if explicitly authorized to do so by the corresponding Web Service protocol specification.

The Web Service Endpoint used is:

`https://<service>.<domain>/.well-known/srv/<service>`

Fallback discovery constrains the service provider to use a specific DNS configuration and provides inferior load balancing or fault tolerance capabilities to use of SRV records. It does however ensure that the service is reachable in situations where it would otherwise be unavailable.

3.6. Example

The Mathematical Mesh has the Well-Known Service name of 'MMM'. Accounts used in the Mathematical Mesh follow the [RFC5322] format of <user>@<domain>.

Alice has the account `alice@example.com` and the DNS configuration file for `example.com` has the following entries:

```
_mmm._tcp.example.com SRV host1.example.com 0 10 80 host1.example.com
_mmm._tcp.example.com SRV host2.example.com 0 40 80 host2.example.com
_mmm._tcp.example.com TXT "version=1.0-2.0"
mmm.example.com       CNAME host3.example.com
host1.example.com     A 10.0.1.1
host2.example.com     A 10.0.1.2
_mmm._tcp.host2.example.com TXT "path=/service"
host3.example.com     A 10.0.1.1
host3.example.com     A 10.0.1.2
```

The client attempts to resolve the address `alice@example.com` as follows:

0. Client attempts to resolve SRV and TXT records for `_mmm._tcp.example.com`
1. DNS resolver returns two SRV entries and one TXT entry
2. Client makes a random selection between `host1` (20% weighting) and `host2` (80% weighting). Chooses `host1`.
3. Client resolves A/AAAA for `host1.example.com` and TXT for `_mmm._tcp.host1.example.com`
4. DNS resolver returns `A=10.0.1.1` and `TXT=none`
5. Client attempts to POST Web Service request to `http://host1example.com/.well-known/srv/mmm` at host address `10.0.1.1`
6. The host at `10.0.1.1` returns 503 Service Unavailable

7. Client resolves A/AAAA for host2.example.com and TXT for _mmm._tcp.host2.example.com
8. DNS resolver returns A=10.0.1.2 and TXT "path=/service"
9. Client attempts to POST Web Service request to http://host2example.com/service at host address 10.0.1.2
10. Request succeeds, session proceeds.

If the same client is used in a network location where the SRV record resolution fails due to a faulty firewall configuration, the resolution proceeds as follows:

0. Client attempts to resolve SRV record for _mmm._tcp.example.com
1. DNS resolver returns 'not found'
2. Client attempts to resolve A and AAAA record
3. DNS resolver returns 10.0.1.1, 10.0.1.2
4. Client makes a random selection between 10.0.1.1 (50% weighting) and 10.0.1.2 (50% weighting). Chooses host1.
5. Client attempts to POST Web Service request to http://example.com/.well-known/srv/mmm at host address 10.0.1.1
6. The host at 10.0.1.1 returns 503 Service Unavailable
7. Client attempts to POST Web Service request to http://example.com/.well-known/srv/mmm at host address 10.0.1.2
8. Request succeeds, session proceeds.

Note that the main differences between these two scenarios is that the use of the SRV record allows the service configuration to account for load balancing with tiers of fallback support and use of service description information while the use of round robin A/AAAA records does not.

4. Further Work

4.1. Additional Description Keys

The use of service and host descriptions to specify security enhancements is currently being considered. This provides a superset of the capabilities specified in [RFC6698].

- * Specify minimum TLS version.
- * Specify trust roots more flexibly
- * Specify client authentication requirements
- * Use of security enhancements other than TLS.
- * Publish public keys to be used to protect negotiation of security enhancements

The use of service and host descriptions to specify use of non-HTTP presentation transports is currently being considered.

4.2. Service Scaling

This document considers the problem of establishing a connection to a Host providing a particular Web Service. When constructing services at very large scale (e.g. millions of concurrent users), it becomes desirable to enable discovery of a Web Service Host responsible for a particular partition of that data (e.g. a particular user account).

Since this is clearly a different problem, it is judged that the best approach is to give it a different name and rule it out of scope of the present work.

5. Security Considerations

A treatment of the security considerations will follow.

6. IANA Considerations

The following registrations are required:

6.1. Well-Known URIs

The following registration is requested in the well-known URI registry in accordance with [RFC5785]

URI suffix

srv

Change controller

Phillip Hallam-Baker, phill@hallambaker.com

Specification document(s):

[This document]

Related information

[draft-hallambaker-web-service-discovery]

7. Normative References

- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <<https://www.rfc-editor.org/rfc/rfc2782>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/rfc/rfc3986>>.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/rfc/rfc5785>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/rfc/rfc6763>>.

8. Informative References

- [draft-hallambaker-web-service-discovery] Hallam-Baker, P., "DNS Web Service Discovery", Work in Progress, Internet-Draft, draft-hallambaker-web-service-discovery-06, 5 August 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-web-service-discovery-06>>.
- [RFC5322] Resnick, P., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <<https://www.rfc-editor.org/rfc/rfc5322>>.
- [RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <<https://www.rfc-editor.org/rfc/rfc6698>>.

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

H. Marques
pEp Foundation
July 02, 2018

pretty Easy privacy (pEp): Email Formats and Protocols
draft-marques-pep-email-00

Abstract

The pretty Easy privacy (pEp) propositions for email are based upon already existing email and encryption formats (i.e., PGP/MIME) and designed to allow for easy implementable and interoperable opportunistic encryption: this ranging from key distribution to mechanisms of subject encryption.

The goal of pEp for email is to automatize operations in order to make email encryption usable by a wider range of Internet users, such that practices for confidentiality and privacy can be achieved in reality.

In this document, basic operations of pEp's approach towards email and two PGP/MIME formats (pEp Email Format 1 and 2) providing certain security guarantees are described.

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terms	4
3. Opportunistic Security with pEp for email	4
3.1. Automatic keypair generation	4
3.2. Key Distribution	5
4. Encryption of email header fields and interoperability	5
5. pEp message formats for email	5
5.1. Unencrypted plain text message with public key attached	5
5.1.1. Example	5
5.2. pEp email format version 1	6
5.2.1. Example 1	7
5.2.2. Example 2	8
5.3. pEp email format version 2	10
5.3.1. Example (Outer and Inner Envelope)	10
6. Rendering Incoming Messages and Message Rating	12
7. Encryption to Bcc recipients	13
7.1. Algorithm	13
7.1.1. Split To and Cc recipients from Bcc recipients	14
7.1.2. Split Bcc recipients in two groups	14
7.1.3. Send one email with only To/Cc recipients	14
7.1.4. Send one Bcc email for the first Bcc group	14
7.1.5. Send individual Bcc emails for the second group	14
8. Saving messages	14
9. Security Considerations	15
10. Implementation Status	15
10.1. Introduction	15
10.2. Current software implementing pEp	15
11. Acknowledgements	16
12. References	16
12.1. Normative References	16
12.2. Informative References	17

Appendix A. Document Changelog	18
Appendix B. Open Issues	18
Author's Address	19

1. Introduction

This document contains specific propositions to those parts of pretty Easy privacy (pEp) [I-D.birk-pep] which are specific to email. [RFC5322]

All changes required for the pEp propositions on email to work just affect implementers of Mail User Agents (MUAs).

pretty Easy privacy (pEp) for email is a proposition to both, implementers and Internet users, to make end-to-end encryption of emails straightforward.

With Pretty Good Privacy (PGP) and OpenPGP [RFC4880] we do not miss the very basis to have good encryption. However, we miss implementatons which make it usable for ordinary Internet users.

Two users using pEp-enabled mail clients basically don't need to do anything else than just writing emails, this working like the following

1. User Alice, knowing nothing of Bob, just writes him an email: this mail goes out unencrypted. However, the Alice's public key is already attached.
2. User Bob can just reply to Alice and is now already able to encrypt a message. Through a color-rating (cf. [I-D.marques-pep-rating] Bob becomes aware of his message now going out in a secure fashion (as secure as the encryption chosen is).
3. User Alice now receives Bob's key in signed and encrypted form and as of now is also able to write secure messages to Bob.
4. If Alice and Bob want to make sure they can exclude a man-in-the-middle attack (MITM), they can engage in a Handshake [I-D.marques-pep-handshake], comparing their so-called Trustwords [I-D.birk-pep-trustwords] and confirm this process if they match. After doing so their identity rating changes to encrypted and authenticated, which (UX-wise) can be done, e.g., using a green color rating. This color rating is also applied to messages (in- and outgoing).

This basic functionality can since longer been shown on different platforms, cf. Section 10).

No propositions are made at this point in time that would require implementers to change the behaviour or feature set of email servers. Another Internet-Draft may propose changes to the Simple Mail Transfer Protocol (SMTP) [RFC5321] as to allow for onion routing of email messages in a way metadata can furtherly be protected for communication peers - achievable by message encapsulation. pEp's email message format 2 described below is already prepared for this scenario.

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].

- o Handshake: The process when Alice - e.g. in-person or via phone - contacts Bob to verify Trustwords (or by fallback: fingerprints) is called handshake. [I-D.marques-pep-handshake]
- o Trustwords: A scalar-to-word representation of 16-bit numbers (0 to 65535) to natural language words. When doing a handshake, peers are shown combined Trustwords of both public keys involved to ease the comparison. [I-D.birk-pep-trustwords]
- o Trust on First Use (TOFU): cf. [RFC7435]
- o Man-in-the-middle attack (MITM): cf. [RFC4949]

3. Opportunistic Security with pEp for email

TBD

3.1. Automatic keypair generation

For every email account a user has in a pEp-enabled Mail User Agent (MUA), a different keypair SHOULD be used by default. If there are no keys whatsoever, RSA-4096 keypairs for OpenPGP encryption [RFC4880] SHOULD be generated automatically for each email account. At the very least RSA-2048 keypairs MUST be generated.

If for an identity there's an RSA keypair with less than 2048 bits, new keys are generated.

3.2. Key Distribution

By default, public keys MUST always be attached to any outgoing message.

4. Encryption of email header fields and interoperability

In pEp, implementers MUST put privacy first: email metadata (i.e., headers) MUST either be omitted or encrypted whenever possible.

In case of email header encryption: implementers of pEp SHOULD be liberal in accepting other approaches to encrypt email headers, but use the strict and interoperable pEp formats for any outgoing communication.

5. pEp message formats for email

With pEp message formats 1 and 2 email security formats are described which are sent signed and encrypted, whenever public key(s) for the recipient(s) exist.

5.1. Unencrypted plain text message with public key attached

If for a recipient there's no public key, a pEp message MUST be sent out in plain text as MIME message version 1, with "Content-Type: multipart/mixed" and the OpenPGP public key attached in ASCII armored format, named "pEpkey.asc".

For a MUA implementer this fulfills two functions:

1. It can easily be detected a pEp user is the sender.
2. The MUA (if at least OpenPGP-enabled) can enable the receiving user to import the public key to engage in end-to-end encryption with the sender; a MUA implementer can also decide to automatically import the key such that the user can immediately engage in opportunistic encryption.

The plain text messages SHOULD be sent out with the UTF-8 charset Content-Type set.

5.1.1. Example

Please note that the "pEpkey.asc" example attachment encoded in base64 format are only shown in its first and last line (and otherwise shortened by three points).

```

From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Test
MIME-Version: 1.0
Content-Type: multipart/mixed;
               boundary="-----3YNFBU8B6LV244ZJNQZL12LVUAPGG6"
Content-Transfer-Encoding: 7bit

```

```

-----3YNFBU8B6LV244ZJNQZL12LVUAPGG6
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain;
charset=UTF-8

```

Test

```

-----3YNFBU8B6LV244ZJNQZL12LVUAPGG6
Content-Type: application/pgp-keys;
  name="pEpkey.asc"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
  filename="pEpkey.asc";
  size=3813

```

```

LS0tLS1CRUdJTiBQRlAgUFVCTElDIETFWSBCTE9DSy0tLS0tCgptUU1OQkZxNWlkd0JF
...
cHhSUXFhQT09CjladlFnCi0tLS0tRU5EIFBHUCBQVUJMSUMgS0VZIEJMT0NLLS0tLS0K

```

```

-----3YNFBU8B6LV244ZJNQZL12LVUAPGG6--

```

5.2. pEp email format version 1

pEp email format 1 is an encrypted and signed PGP/MIME format, which by default ensures

- o correctly signed messages
- o delivery of public keys (at least and automatically: the sender's public key)

By default, when a public key for a peer is available, pEp-capable MUAs are REQUIRED to send out email messages according to [RFC5322] and in PGP/MIME format [RFC3156] with the informational "Subject:" header field set to "pEp", as follows:

```
Subject: pEp
```

In turn, the intended human-readable subject (in pEp called short message) MUST be moved to the body of the message (in pEp called long

message) and appear as the first line there. pEp implementers are REQUIRED to display the intended "Subject:" field as the real subject line in the respective MUAs to help users to easily grasp the real subject.

The "Subject:" header field can also be set to its UTF-8 variant with "pEp" written with the equivalence symbol instead of an "E":

```
Subject: =?utf-8?Q?p=E2=89=Alp?=
```

Additionally, a header field "X-Pep-Version: 1.0" is to be added as to make clear a user is using a pEp-enabled MUA with pEp email format 1.

5.2.1. Example 1

Example. Using the well-known example of [RFC5322], an email message sent out with pEp in message format 1 looks like this:

From: John Doe <jdoe@machine.example>
Sender: Michael Jones <mjones@machine.example>
To: Mary Smith <mary@example.net>
Subject: pEp
Date: Fri, 30 Jun 2018 09:55:06 +0200
Message-ID: <1234@local.machine.example>
MIME-Version: 1.0
Content-Type: multipart/mixed;
 boundary="-----_NextPart_000_0016_01D0E64A.33EC31B0"
Content-Language: en-us
X-Pep-Version: 1.0

This is a multipart message in MIME format.

-----=_NextPart_000_0016_01D0E64A.33EC31B0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 7bit

-----BEGIN PGP MESSAGE-----
hQIMAwusnBHN80H+AQ//cJLQLOl+6hOofKEkQJeu0wedmwt+TkzPx/sCUQ80dzLv
...
j/ES8ndDBftM5mZLzFQ2VatqB9G9cqCgiOVFs6jfTI13nPfLit9IPWRavcVIMdwt
Xd9bdvHx/ReenAk/
=7WaL
-----END PGP MESSAGE-----

-----=_NextPart_000_0060_01D0EAEF.2D54F450
Content-Type: application/pgp-keys; name="pEp_key.asc"
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="pEp_key.asc"

-----BEGIN PGP PUBLIC KEY BLOCK-----
mQINBFQRqIcBEACpsz3mK1zqPdqDlxU6Yws/Xz14LJpsZDLlKJckpa7hSc9jffZ4Q
...
Ag7IIk/Gj628hYTdCpNCUc9blvS6xMAkxJWYgNVwLFS2goikeEHCIyzDe
=MicJ
-----END PGP PUBLIC KEY BLOCK-----

-----=_NextPart_000_0060_01D0EAEF.2D54F450--

5.2.2. Example 2

Using the UTF-8 variant of writing "pEp" with the equivalence symbol, and an additional document attached (an example PDF attachment), an OpenPGP-signed and -encrypted pEp email would look like the following:

From: John Doe <jdoe@machine.example>
Sender: Michael Jones <mjones@machine.example>
To: Mary Smith <mary@example.net>
Subject: =?utf-8?Q?p=E2=89=A1p?=
Date: Fri, 30 Jun 2018 09:55:06 +0200
Message-ID: <1234@local.machine.example>
MIME-Version: 1.0
Content-Type: multipart/mixed;
 boundary="-----_NextPart_000_0016_01D0E64A.33EC31B0"
Content-Language: en-us
X-Pep-Version: 1.0

This is a multipart message in MIME format.

-----=_NextPart_000_0016_01D0E64A.33EC31B0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 7bit

-----BEGIN PGP MESSAGE-----
hQIMAwusnBHN80H+AQ//cJLQLOl+6hOofKEkQJeu0wedmwt+TkzPx/sCUQ80dzLv
...
j/ES8ndDBftM5mZLzFQ2VatqB9G9cqCgiOVFs6jftI13nPfLit9IPWRavcVIMdwt
Xd9bdvHx/ReenAk/
=7WaL
-----END PGP MESSAGE-----

-----=_NextPart_000_003A_01D10CF6.2DA15150
Content-Type: application/octet-stream; name="example.pdf.pgp"
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="example.pdf.pgp"

-----BEGIN PGP MESSAGE-----
hQIMA/bohV/mG7k7ARAAyy+sdpZYZBhUH/p0gJ+wIleGTTG2rjLpLuixBrm5Cuj3
...
oAXrQJjgD0F3Ung24Kkundua2gSa9cyeYvUXtA2mbXT7YyN7RdxrMFNfdVFqXZEc
pXqIjL2uKBbyjpS44fc3GmOZNih3bi6q8nl/
=Mvna

-----=_NextPart_000_0060_01D0EAEF.2D54F450
Content-Type: application/pgp-keys; name="pEp_key.asc"
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="pEp_key.asc"

-----BEGIN PGP PUBLIC KEY BLOCK-----
mQINBFQRqIcBEACpsz3mK1zqPdqDlxU6Yws/Xz14LJpszDLlKJckpa7hSc9jffZ4Q
...
Ag7IIk/Gj628hYtdCpNCUc9blvS6xMAkxJWYgNVwLFS2goikeEHciyzDe
=MicJ

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

-----=_NextPart_000_0060_01D0EAEF.2D54F450--

5.3. pEp email format version 2

pEp email format 2 is a strict PGP/MIME format, which by default ensures

- o correctly signed messages
- o delivery of public keys (at least: the sender's public key)

In pEp email format 2 the actual email is encapsulated by an outside multipartd/encrypted envelope email (i.e., the actual email is sent like a forwarded message).

Headers of messages (received, to be forwarded etc.) can thus be preserved in the inner message, which is OpenPGP-signed and -encrypted by the application/pgp-encrypted "Content-Type:".

In the outer envelope, unnecessary email headers MUST be omitted to the fullest extent.

In contrast to pEp email format 1, the public key and other files attached cannot be seen in the MIME tree. The only part which can be seen is an application/octet-stream "Content-Type" with name "msg.asc".

or the sender's public key is considered as modification and shown as attack.

5.3.1. Example (Outer and Inner Envelope)

A pEp email format 2 message, with the "Subject:" header field set to "pEp" looks like the following; please note that the inner envelope is fully contained in the OpenPGP-signed and -encrypted file named "msg.asc", including possible attachments and with the sender's public key as "pEpkey.asc" attached at the very end:

From: John Doe <jdoe@machine.example>
Sender: Michael Jones <mjones@machine.example>
To: Mary Smith <mary@example.net>
Subject: =?utf-8?Q?p=E2=89=Alp?=
Date: Fri, 30 Jun 2018 09:55:06 +0200
Message-ID: <1234@local.machine.example>
MIME-Version: 1.0
Subject: pEp
X-Pep-Version: 2.0
Content-Type: multipart/encrypted;
 boundary="261a304d18692673570d913f7e24b8cb";
 protocol="application/pgp-encrypted"

--261a304d18692673570d913f7e24b8cb
Content-Type: application/pgp-encrypted

Version: 1
--261a304d18692673570d913f7e24b8cb
Content-Type: application/octet-stream
Content-Transfer-Encoding: 7bit
Content-Disposition: inline; filename="msg.asc"

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

hQGMAzDKu5MiiyCzAQv9Edg8ulxgxyQfiZRxOpThL0aMFkK7JZH7AJfgdxunLAJk
...
a2jDdzNxotItZk8tWW2h/REdKtRMxYg633DyFLbsIx+cCMnMR1NDChCzvyzUjAw6
XeCGXnY3LB1K
=sdgE
-----END PGP MESSAGE-----

--261a304d18692673570d913f7e24b8cb--

The inner envelope in a simple form without further nestings might
look like the following, when decrypted:

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="17d3c87b380049a821c764604aaf9272"

--17d3c87b380049a821c764604aaf9272
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: quoted-printable
Content-Disposition: inline; filename="msg.txt"

Subject: The real encrypted subject
```

Hello, there!

```
--17d3c87b380049a821c764604aaf9272
Content-Type: application/pgp-keys
Content-Disposition: attachment; filename="pEpkey.asc"
```

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
mQGNBFmwE70BDACyR/yQ48QSaQAZyvyUgp7f/4WXxiX1OS9vC/UuewdGLosvl3G+
...
A0KQ6HDwLFuLzneg6Nse4pX0hNWGbLNCouYKdL3vfUHokqp/MTzxyPQlOadDHRDV
H9RC4kMrB/ONGe5yn+u4zjrgq9gWCbdJ43fMoiU3lfMIKy5sZ2NPzh9l
=p5bZ
-----END PGP PUBLIC KEY BLOCK-----
```

It does not only carry the encrypted subject, which pEp implementers are supposed to map (UX-wise) such as to replace the "pEp" subject in the outer envelope, but also the actual message (as inline file named "msg.txt" in case of plain text) as well as the the sender's public key.

6. Rendering Incoming Messages and Message Rating

pEp-enabled clients MUST NOT blindly render messages. Special care MUST be taken when rendering the pEp email formats, which provide certain guarantees:

Message Format	Error State	Render	Status Code
PGP/MIME	Unsigned	Yes	DECRYPTED_BUT_UNSIGNED
	Signed, no key	Yes	NO_KEY_FOR_SIGNER
	Bad signature	No	SIGNATURE_DOES_NOT_MATCH
pEp Email 1.0	Unsigned	No	DECRYPTED_BUT_UNSIGNED
	Signed, no key	No	NO_KEY_FOR_SIGNER
	Bad signature	No	SIGNATURE_DOES_NOT_MATCH
pEp Email 2.0	Unsigned	No	MODIFICATION_DETECTED
	Signed, no key	No	MODIFICATION_DETECTED
	Bad signature	No	SIGNATURE_DOES_NOT_MATCH

For cases where messages appear unsigned, signed without a key or with a bad signature, pEp's privacy rating can be employed to signal issues to a user in an easily understandable manner, cf. [I-D.marques-pep-rating].

[[TODO: This needs more work to be understandable.]]

7. Encryption to Bcc recipients

7.1. Algorithm

For email encryption including Bcc recipients, which MUST receive encrypted messages if public keys are available for any of the involved email addresses, this simple algorithm MAY be used.

Recipients MUST be partitioned into three lists, one for each of three possible outgoing messages:

1. To and Cc recipients without Bcc recipients.
2. Bcc recipients unable to encrypt.
3. Bcc recipients able to encrypt.

It's RECOMMENDED that if the original message the user drafted is saved in the user's sent folder, that all recipient fields ("To:", "Cc:", "Bcc:") be preserved.

7.1.1. Split To and Cc recipients from Bcc recipients

To and Cc recipients MUST be split from the Bcc recipients.

7.1.2. Split Bcc recipients in two groups

Bcc recipients MUST be split in two groups:

- o First group of Bcc recipients who will receive clear text emails.
- o Second group of Bcc recipients who are able to receive encrypted emails.

7.1.3. Send one email with only To/Cc recipients

The original email the user drafted SHOULD be sent out with the "Bcc:" field removed.

7.1.4. Send one Bcc email for the first Bcc group

For the first Bcc group, a regular email message with only Bcc recipients is sent.

7.1.5. Send individual Bcc emails for the second group

For the second group, individual Bcc email messages are sent.

8. Saving messages

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

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

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

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

For trusted environments (e.g., in organizations) and to conform to legally binding regulations, pEp implementations MUST provide a "Trusted Server" option. With the user's explicit consent (opt-in), unencrypted copies of the messages SHALL be held on the mail servers controlled by the organization. This can also help end-users to archive their emails without needing access to any key material.

9. Security Considerations

[[TODO]]

10. Implementation Status

10.1. 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."

10.2. Current software implementing pEp

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

- o pEp for Outlook as add-on for Microsoft Outlook, release [SRC. pepforoutlook]
- o pEp for Android (based on a fork of the K9 MUA), release [SRC. pepforandroid]
- o Enigmail/pEp as add-on for Mozilla Thunderbird, release [SRC. enigmailpep]

- o pEp for iOS (implemented in a new MUA), beta [SRC.pepforios]

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.

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

11. Acknowledgements

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

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

12. References

12.1. Normative References

- [I-D.birk-pep] Birk, V., Marques, H., and S. Shelburn, "pretty Easy privacy (pEp): Privacy by Default", draft-birk-pep-02 (work in progress), June 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3156] Elkins, M., Del Torto, D., Levien, R., and T. Roessler, "MIME Security with OpenPGP", RFC 3156, DOI 10.17487/RFC3156, August 2001, <<https://www.rfc-editor.org/info/rfc3156>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.
- [RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <<https://www.rfc-editor.org/info/rfc5322>>.

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

12.2. Informative References

- [I-D.birk-pep-trustwords]
Birk, V., Marques, H., and B. Hoeneisen, "IANA Registration of Trustword Lists: Guide, Template and IANA Considerations", draft-birk-pep-trustwords-02 (work in progress), June 2018.
- [I-D.marques-pep-handshake]
Marques, H. and B. Hoeneisen, "pretty Easy privacy (pEp): Contact and Channel Authentication through Handshake", draft-marques-pep-handshake-00 (work in progress), June 2018.
- [I-D.marques-pep-rating]
Marques, H. and B. Hoeneisen, "pretty Easy privacy (pEp): Mapping of Privacy Rating", draft-marques-pep-rating-00 (work in progress), July 2018.
- [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/>>.
- [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>>.
- [RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <<https://www.rfc-editor.org/info/rfc5321>>.
- [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>>.

[SRC.enigmailpep]
"Source code for Enigmail/pEp", July 2018,
<<https://enigmail.net/index.php/en/download/source-code>>.

[SRC.pepforandroid]
"Source code for pEp for Android", July 2018,
<<https://pep-security.lu/gitlab/android/pep>>.

[SRC.pepforios]
"Source code for pEp for iOS", July 2018,
<https://pep-security.ch/dev/repos/pEp_for_iOS/>.

[SRC.pepforoutlook]
"Source code for pEp for Outlook", July 2018,
<https://pep-security.lu/dev/repos/pEp_for_Outlook/>.

Appendix A. Document Changelog

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

- o draft-marques-pep-email-00:

- * Initial version

Appendix B. Open Issues

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

- o Ship better example of pEp Message Format 2
- o Elaborate on omitting headers and better explain pEp Message Format 2
- o Add notes on EFAIL
- o Describe KeyImport to induce the import from secret keys from other devices
- o Describe / Reference KeySync (and other sync, through IMAP)
- o Add keypair revocation strategy
- o Better describe required MIME fields and parameters to set for the pEp email formats
- o Create clearer relations to the pEp rating draft (draft-marques-pep-rating), as this plays an important role in how messages are

rendered and how they need to be presented (after rating) for a user to have awareness about his privacy status in any given situation.

- o Make document more coherent: check with pEp's general draft pieces to fill on both sides and how to reference them vice-versa.

Author's Address

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

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 1, 2019

H. Marques
pEp Foundation
B. Hoeneisen
Ucom.ch
June 30, 2018

pretty Easy privacy (pEp): Contact and Channel Authentication through
Handshake
draft-marques-pep-handshake-00

Abstract

In interpersonal messaging end-to-end encryption means for public key distribution and verification of its authenticity are needed; the latter to prevent man-in-the-middle (MITM) attacks.

This document proposes a new method to easily verify a public key is authentic by a Handshake process that allows users to easily verify their communication channel. The new method is targeted to Opportunistic Security scenarios and is already implemented in several applications of pretty Easy privacy (pEp).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 1, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terms	3
3. Problem Statement	3
3.1. Use Cases	4
3.2. Existing Solutions	4
4. The pEp Handshake Proposal	5
4.1. Verification Process	5
4.1.1. Partial and Full Trustword Mapping	6
4.1.2. Display Modes	7
5. Security Considerations	7
6. Implementation Status	8
6.1. Introduction	8
6.2. Running Code	8
7. Acknowledgements	8
8. References	9
8.1. Normative References	9
8.2. Informative References	9
Appendix A. Open Issues	11
Authors' Addresses	11

1. Introduction

In interpersonal messaging end-to-end encryption means for public key distribution and verification of its authenticity are needed.

Examples for key distribution include:

- o Exchange keys out-of-band before encrypted communication
- o Use of centralized public key stores (e.g., OpenPGP Key Servers)
- o Ship the public key in-band when communicating

To prevent man-in-the-middle (MITM) attacks, additionally the authenticity of a public key needs to be verified. Methods to authenticate public keys of peers include, e.g., verify digital signatures of public keys (which may be signed in an hierarchical or flat manner, e.g., by a Web of Trust (WoT)), compare the public key's

fingerprints via a suitable independent channel, or scan a QR mapping of the fingerprint (cf. Section 3).

This document proposes a new method to verify the authenticity of public keys by a Handshake process that allows users to easily verify their communication channel. Fingerprints of the involved peers are combined and mapped to (common) Trustwords [I-D.birk-pep-trustwords]. The successful manual comparison of these Trustwords is used to consider the communication channel as trusted.

The proposed method is already implemented and used in applications of pretty Easy privacy (pEp) [I-D.birk-pep]. This document is targeted to applications based on the pEp framework and Opportunistic Security [RFC7435]. However, it may be also used in other applications as suitable.

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].

- o Handshake: The process when Alice - e.g. in-person or via phone - contacts Bob to verify Trustwords (or by fallback: fingerprints) is called Handshake. In more detail, this is described in this draft.
- o Trustwords: A scalar-to-word representation of 16-bit numbers (0 to 65535) to natural language words. When doing a Handshake, peers are shown combined Trustwords of both public keys involved to ease the comparison. [I-D.birk-pep-trustwords]
- o Trust on First Use (TOFU): cf. [RFC7435]
- o Man-in-the-middle attack (MITM): cf. [RFC4949]

3. Problem Statement

To secure a communication channel, in public key cryptography the each involved peer needs a key pair. Its public key needs to be shared to other peers by some means. However, the key obtained by the receiver may have been substituted or tampered with to allow for re-encryption attacks. To prevent such man-in-the-middle (MITM) attacks, an important step is to verify the authenticity of a public key obtained.

3.1. Use Cases

Such a verification process is useful in at least two scenarios:

- o Verify channels to peers, e.g., to make sure opportunistically exchanged keys for end-to-end encryption are authentic.
- o Verify channels between own devices (in multi-device contexts), e.g., for the purpose of importing and synchronizing keys among different devices belonging to the same user (cf. [E-D.birk-pep-keysync]). This scenario is comparable to Bluetooth pairing before starting data transfers.

3.2. Existing Solutions

Current methods to authenticate public keys of peers include:

- o Digitally signed public keys are verified by a chain of trust. Two trust models are common in today's implementations.
 - * Signing is carried out hierarchically, e.g. in a Public Key Infrastructure (PKI) [RFC5280], in which case the verification is based on a chain of trust with a Trust Anchor (TA) at the root.
 - * Signing of public keys is done in a flat manner (by a Web of Trust), e.g., key signing in OpenPGP [RFC4880], where users sign the public keys of other users. Verification may be based on transitive trust.
- o Peers are expected to directly compare the public key's fingerprints by any suitable independent channel - e.g, by phone or with a face-to-face meeting. This method is often used in OpenPGP [RFC4880] contexts.
- o The public keys' fingerprints are mapped into a QR code, which is expected to be scanned between the peers when they happen to meet face-to-face. This method is e.g. used in the chat application Threema [threema].
- o The public keys' fingerprints are mapped into numerical codes which decimal digits only, which makes the strings the humans need to compare longer and thus cumbersome. This method is e.g. used in the chat application Signal [signal].

Some of the methods can even be used in conjunction with Trustwords [I-D.birk-pep-trustwords] or the PGP Word list [PGP.wl].

None of the existing solutions meets all requirements set up by pEp [I-D.birk-pep], e.g.:

- o Easy solution that can be handled easily by ordinary users
- o In case privacy and security contradict each other, privacy is always preferred over security (e.g., the Web of Trust contradicts privacy)
- o No central entities

Most of today's systems lack easy ways for users to authenticate their communication channel. Some methods leak private data (e.g., their social graph) or depend on central entities. Thus, none of today's systems fulfills all of the pEp requirements (cf. above).

4. The pEp Handshake Proposal

In pretty Easy privacy (pEp), the proposed approach for peers to authenticate each other is to engage in the pEp Handshake.

In current pEp implementations (cf. Section 6.2), the same kinds of keys as in OpenPGP are used. Such keys include a fingerprint as cryptographic hash over the public key. This fingerprint is normally represented in a hexadecimal form, consisting of ten 4-digit hexadecimal blocks, e.g.:

8E31 EF52 1D47 5183 3E9D EADC 0FFE E7A5 7E5B AD19

Each block may also be represented in decimal numbers from 0 to 65535 or in other numerical forms, e.g.

- o Hexadecimal: 8E31
- o Decimal: 36401
- o Binary: 1000111000110001

4.1. Verification Process

In the pEp Handshake the fingerprints of any two peers involved are combined and displayed in form of Trustwords [I-D.birk-pep-trustwords] for easy comparison by the involved parties.

The default verification process involves three steps:

1. Combining fingerprints by XOR function

Any two peers' fingerprints are combined bit-by-bit using the Exclusive-OR (XOR) function resulting in the Combined Fingerprint (CFP).

2. Mapping result to Trustwords:

The CFP is then mapped to 16-bit Trustwords (i.e., every 4-digit hexadecimal block is mapped to a given Trustword) resulting in the Trustword Mapping (TWM).

3. Verify Trustwords over independent channel

The resulting Trustwords (TWM) are compared and verified over an independent channel, e.g., a phone line. If this step was successful, the channel can be marked as authenticated.

4.1.1. Partial and Full Trustword Mapping

The more an ordinary user needs to contribute to a process, the less likely a user will carry out all steps carefully. In particular, it was observed that a simple (manual) comparison of OpenPGP fingerprints is rarely carried out to the full extent, i.e., mostly only parts of the fingerprint are compared, if at all.

For usability reasons and to create incentive for people to actually carry out Handshake (while maintaining an certain level of entropy), pEp allows for different entropy levels, i.e.:

1. Full Trustword Mapping (F-TWM) [aka Full Trustwords] MUST represent the maximum entropy achievable by the mapping. This means all Trustwords of a TWM MUST be displayed and compared.

e.g. the fingerprint

F482 E952 2F48 618B 01BC 31DC 5428 D7FA ACDC 3F13

is mapped to:

dog house brother town fat bath school banana kite task

2. Partial Trustword Mapping (P-TWM) [aka Short Trustwords] requires a number of Trustwords that MUST retain at least 64-bit of entropy. This means while the first part of the TWM is displayed and compared, the remaining Trustwords are omitted.

e.g. the fingerprint

F482 E952 2F48 618B 01BC [31DC 5428 D7FA ACDC 3F13]

is mapped to:

dog house brother town fat [remaining Trustwords omitted]

4.1.2. Display Modes

The pEp Handshake has three display modes for the verification process. All of the following modes MUST be implemented:

1. P-TWM mode (default)

By default the P-TWM SHOULD be displayed to the user for comparison and verification. An easy way to switch to F-TWM mode MUST be implemented and an easy way to switch to fingerprint mode SHOULD be implemented.

2. F-TWM mode

There are situations, where P-TWM is not sufficient (e.g., communication parties that are more likely under attack), in which the F-TWM MAY be displayed to the user by default. An easy way to switch to P-TWM mode MUST be implemented and an easy way to switch to fingerprint mode SHOULD be implemented.

3. Fingerprint mode (fallback)

To retain compatibility to existing OpenPGP users (that know nothing about Trustwords), the fingerprint mode, a fallback to compare the original fingerprints (usually in hexadecimal form) MAY be used. Easy ways to switch to P-TWM and F-TWM modes SHOULD be implemented.

If the verification process was successful, the user confirms it, e.g. by setting a check mark. Once the user has confirmed it, the trust level [E-D.birk-pep-trust-rating] for this channel MUST be updated accordingly.

5. Security Considerations

A (global) adversary can pre-generate all Trustwords any two users expect to compare and try to engage in MITM attacks which fit - it MUST NOT be assumed public keys and thus fingerprints to be something to stay secret, especially as in pEp public keys are aggressively distributed to all peers. Also similar Trustwords can be generated, which spelled on the phone might sound very similar.

6. Implementation Status

6.1. 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."

6.2. Running Code

In pEp for email [E-D.birk-pep-email] contexts, Handshakes are already implemented for the following platforms:

- o Android, in pEp for Android - release [SRC.pepforandroid]
- o Enigmail, in the Enigmail/pEp mode - release used for new Enigmail users of version 2.0 [SRC.enigmailpep]
- o iOS, in pEp for iOS - not yet released [SRC.pepforios]
- o Outlook, in pEp for Outlook - commercial release [SRC.pepforoutlook]

In pEp for Outlook also keys from other devices can be imported by the Handshake method.

7. Acknowledgements

Special thanks to Volker Birk and Leon Schumacher who developed the original concept of the pEp Handshake.

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]

8. References

8.1. Normative References

[I-D.birk-pep]

Birk, V., Marques, H., and S. Shelburn, "pretty Easy privacy (pEp): Privacy by Default", draft-birk-pep-02 (work in progress), June 2018.

[I-D.birk-pep-trustwords]

Birk, V., Marques, H., and B. Hoeneisen, "IANA Registration of Trustword Lists: Guide, Template and IANA Considerations", draft-birk-pep-trustwords-02 (work in progress), June 2018.

[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>>.

8.2. Informative References

[E-D.birk-pep-email]

Birk, V. and H. Marques, "pretty Easy privacy (pEp): Secure and Trusted Email Communication", June 2018, <<https://pep.foundation/dev/repos/internet-drafts/file/tip/pep-email/draft-birk-pep-email-NN.txt>>.

Early draft

[E-D.birk-pep-keystsync]

Birk, V. and H. Marques, "pretty Easy privacy (pEp): Key Synchronization Protocol", June 2018, <<https://pep.foundation/dev/repos/internet-drafts/file/tip/pep-keystsync/draft-birk-pep-keystsync-NN.txt>>.

Early draft

[E-D.birk-pep-trust-rating]

Birk, V. and H. Marques, "pretty Easy privacy (pEp): Trust Rating System", June 2018, <<https://pep.foundation/trac/browser/internet-drafts/pep-rating/draft-marques-pep-rating-00.txt>>.

Early draft

[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/>>.

[PGP.wl]

"PGP word list", November 2017, <https://en.wikipedia.org/w/index.php?title=PGP_word_list&oldid=749481933>.

[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>>.

[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>>.

[signal]

"Signal", n.d., <<https://signal.org/>>.

[SRC.enigmailpep]

"Source code for Enigmail/pEp", June 2018, <<https://enigmail.net/index.php/en/download/source-code>>.

[SRC.pepforandroid]

"Source code for pEp for Android", June 2018, <<https://pep-security.lu/gitlab/android/pep>>.

[SRC.pepforios]

"Source code for pEp for iOS", June 2018,
<https://pep-security.ch/dev/repos/pEp_for_iOS/>.

[SRC.pepforoutlook]

"Source code for pEp for Outlook", June 2018,
<https://pep-security.lu/dev/repos/pEp_for_Outlook/>.

[threema]

"Threema - Seriously secure messaging", n.d.,
<<https://threema.ch>>.

Appendix A. Open Issues

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

- o Add description for further processes to change the trust level,
e.g., to remove trust or even mistrust a peer and alike.

Authors' Addresses

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

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

Bernie Hoeneisen
Ucom Standards Track Solutions GmbH
CH-8046 Zuerich
Switzerland

Phone: +41 44 500 52 44
Email: bernie@ietf.hoeneisen.ch (bernhard.hoeneisen AT ucom.ch)
URI: <https://ucom.ch/>

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

H. Marques
pEp Foundation
B. Hoeneisen
Ucom.ch
July 02, 2018

pretty Easy privacy (pEp): Mapping of Privacy Rating
draft-marques-pep-rating-00

Abstract

In many Opportunistic Security scenarios end-to-end encryption is automatized for Internet users. In addition, it is often required to provide the users with easy means to carry out authentication.

Depending on several factors, each communication channel to different peers may have a different Privacy Status, e.g., unencrypted, encrypted and encrypted as well as authenticated. Even each message from/to a single peer may have a different Privacy Status.

To display the actual Privacy Status to the user, this document defines a Privacy Rating scheme and its mapping to a traffic-light semantics. A Privacy Status is defined on a per-message basis as well as on a per-identity basis. The traffic-light semantics (as color rating) allows for a clear and easily understandable presentation to the user in order to optimize the User Experience (UX).

This rating system is most beneficial to Opportunistic Security scenarios and is already implemented in several applications of pretty Easy privacy (pEp).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terms	4
3. Per-Message Privacy Rating	4
3.1. Rating Codes	4
3.2. Color Codes	5
3.3. Surjective Mapping of Rating Codes into Color Codes . . .	6
3.4. Semantics of Color and Rating Codes	6
3.4.1. Red	6
3.4.2. No Color	6
3.4.3. Yellow	7
3.4.4. Green	7
4. Per-Identity Privacy Rating	7
5. Security Considerations	8
6. Implementation Status	8
6.1. Introduction	8
6.2. Running Code	9
7. Acknowledgments	9
8. References	9
8.1. Normative References	9
8.2. Informative References	10
Appendix A. Excerpts from the pEp Reference Implementation . . .	11
A.1. pEp rating	11
Appendix B. Document Changelog	11
Appendix C. Open Issues	11
Authors' Addresses	12

1. Introduction

In many Opportunistic Security [RFC7435] scenarios end-to-end encryption is automatized for Internet users. In addition, it is often required to provide the users with easy means to carry out authentication.

Depending on several factors, each communication channel to different identities may have a different Privacy Status, e.g.

- o unreliable
- o encrypted
- o encrypted and authenticated
- o mistrusted

Even each message from or to a single peer may have a different Privacy Status.

To display the actual Privacy Status to the user, this document defines a Privacy Rating scheme and its mapping to a traffic-light semantics, i.e., a mapping to different color codes as used in a traffic-light:

- o red
- o yellow
- o green
- o no color (or gray)

Note: While "yellow" color is used in the context of traffic-lights (e.g., in North America), in other parts of the world (e.g., the UK) this is generally referred to as "orange" or "amber" lights. For the scope of this document, "yellow", "amber", and "orange" refer to the same semantics.

A Privacy Status is defined on a per-message basis as well as on a per-identity basis. The traffic-light semantics (as color rating) allows for a clear and easily understandable presentation to the user in order to optimize the User Experience (UX). To serve also (color-)blind Internet users or those using monochrome displays, the traffic light color semantics may also be presented by simple texts and symbols for signaling the corresponding Privacy Status.

The proposed definitions are already implemented and used in applications of pretty Easy privacy (pEp) [I-D.birk-pep]. This document is targeted to applications based on the pEp framework and Opportunistic Security [RFC7435]. However, it may be also used in other applications as suitable.

Note: The pEp [I-D.birk-pep] framework proposes to automatize the use of end-to-end encryption for Internet users of email and other messaging applications and introduces methods to easily allow authentication.

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].

- o Handshake: The process when Alice - e.g., in-person or via phone - contacts Bob to verify Trustwords (or by fallback: fingerprints) is called handshake. [I-D.marques-pep-handshake]
- o Trustwords: A scalar-to-word representation of 16-bit numbers (0 to 65535) to natural language words. When doing a handshake, peers are shown combined Trustwords of both public keys involved to ease the comparison. [I-D.birk-pep-trustwords]
- o Trust on First Use (TOFU): cf. [RFC7435]
- o Man-in-the-middle attack (MITM): cf. [RFC4949]

3. Per-Message Privacy Rating

3.1. Rating Codes

To rate messages (cf. also Appendix A.1), the following 13 Rating codes are defined as scalar values (decimal):

Rating code	Rating label
-3	under attack
-2	broken
-1	mistrust
0	undefined
1	cannot decrypt
2	have no key
3	unencrypted
4	unencrypted for some
5	unreliable
6	reliable
7	trusted
8	trusted and anonymized
9	fully anonymous

3.2. Color Codes

For an Internet user to understand what the available Privacy Status is, the following colors (traffic-light semantics) are defined:

Color code	Color label
-1	red
0	no color
1	yellow
2	green

3.3. Surjective Mapping of Rating Codes into Color Codes

Corresponding User Experience (UX) implementations use a surjective mapping of the Rating Codes into the Color Codes (in traffic light semantics) as follows:

Rating codes	Color code	(Color label)
-3 to -1	-1	(red)
0 to 5	0	(no color)
6	1	(yellow)
7 to 9	2	(green)

This mapping is used in current pEp implementations to signal the Privacy Status (cf. Section 6.2).

3.4. Semantics of Color and Rating Codes

3.4.1. Red

The red color MUST only be used in three cases:

- o Rating code -3: A man-in-the-middle (MITM) attack could be detected.
- o Rating code -2: The message was tempered with.
- o Rating code -1: The user explicitly states he mistrusts a peer, e.g., because a Handshake [I-D.marques-pep-handshake] mismatched or when the user learns the communication partner was attacked and might have gotten the corresponding secret key leaked.

3.4.2. No Color

No specific (or a gray color) MUST be shown in the following cases:

- o Rating code 0: A message can be rendered, but the encryption status is not clear, i.e., undefined
- o Rating code 1: A message cannot be decrypted (because of an error not covered by rating code 2 below).

- o Rating code 2: No key is available to decrypt a message (because it was encrypted with a public key for which no secret key could be found).
- o Rating code 3: A message is received or sent out unencrypted (because it was received unencrypted or there's no public key to encrypt a message to a recipient).
- o Rating code 4: A message is sent out unencrypted for some of the recipients of a group (because there's at least one recipient in the group whose public key is not available to the sender).
- o Rating code 5: A message is encrypted, but cryptographic parameters (e.g., the cryptographic method employed or key length) are insufficient.

3.4.3. Yellow

- o Rating code 6: Whenever a message can be encrypted or decrypted with sufficient cryptographic parameters, it's considered reliable. It is mapped into the yellow color code.

3.4.4. Green

- o Rating code 7: A message is mapped into the green color code only if a pEp handshake [I-D.marques-pep-handshake] was successfully carried out.

By consequence that means, that the pEp propositions don't strictly follow the TOFU (cf. [RFC7435]) approach, in order to avoid signaling trust without peers verifying their channel first.

In current pEp implementations (cf. Section 6) only rating code 7 is achieved.

The rating codes 8 and 9 are reserved for future use in pEp implementations which also secure meta-data (rating code 8), by using a peer-to-peer framework like GNUnet [GNUnet], and/or allow for fully anonymous communications (rating code 9), where sender and receiver don't know each other, but trust between the endpoints could be established nevertheless.

4. Per-Identity Privacy Rating

The same Color Codes (red, no color, yellow and green) as for messages (cf. Section 3.2) MUST be applied for identities (peers), so that a user can easily understand, which identities private communication is possible with.

The green color code MUST be applied to an identity whom the pEp handshake [I-D.marques-pep-handshake] was successfully carried out with.

The yellow color code MUST be set whenever a public key could be obtained to securely encrypt messages to an identity, although a MITM attack cannot be excluded.

The no color code MUST be used for the case that no public key is available to engage in private communications with an identity.

The red color code MUST only be used when an identity is marked as mistrusted.

[[It's not yet clear if there are proper cases where it makes sense to set an identity automatically to the red color code, as it appears to be difficult to detect attacks (e.g., secret key leakage) at the other endpoint with certainty.]]

5. Security Considerations

[[TODO]]

6. Implementation Status

6.1. 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."

6.2. Running Code

In pEp for email contexts, pEp rating codes are already implemented for the following platforms:

- o Android, in pEp for Android - release [SRC.pepforandroid]
- o Enigmail, in the Enigmail/pEp mode - release used for new Enigmail users of version 2.0 [SRC.enigmailpep]
- o iOS, in pEp for iOS - not yet released [SRC.pepforios]
- o Outlook, in pEp for Outlook - commercial release [SRC.pepforoutlook]

7. Acknowledgments

The authors would like to thank the following people who have provided feedback or significant contributions to the development of this document: Leon Schumacher and Volker Birk

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]

8. References

8.1. Normative References

- [I-D.birk-pep] Birk, V., Marques, H., and S. Shelburn, "pretty Easy privacy (pEp): Privacy by Default", draft-birk-pep-02 (work in progress), June 2018.
- [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>>.

8.2. Informative References

- [GNUnet] Grothoff, C., "The GNUnet System", October 2017, <<https://grothoff.org/christian/habil.pdf>>.
- [I-D.birk-pep-trustwords] Birk, V., Marques, H., and B. Hoeneisen, "IANA Registration of Trustword Lists: Guide, Template and IANA Considerations", draft-birk-pep-trustwords-02 (work in progress), June 2018.
- [I-D.marques-pep-handshake] Marques, H. and B. Hoeneisen, "pretty Easy privacy (pEp): Contact and Channel Authentication through Handshake", draft-marques-pep-handshake-00 (work in progress), June 2018.
- [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/>>.
- [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>>.
- [SRC.enigmailpep] "Source code for Enigmail/pEp", July 2018, <<https://enigmail.net/index.php/en/download/source-code>>.
- [SRC.pepforandroid] "Source code for pEp for Android", July 2018, <<https://pep-security.lu/gitlab/android/pep>>.
- [SRC.pepforios] "Source code for pEp for iOS", July 2018, <https://pep-security.ch/dev/repos/pEp_for_iOS/>.
- [SRC.pepforoutlook] "Source code for pEp for Outlook", July 2018, <https://pep-security.lu/dev/repos/pEp_for_Outlook/>.

Appendix A. Excerpts from the pEp Reference Implementation

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

A.1. pEp rating

From the reference implementation by the pEp foundation, src/message_api.h:

```
typedef enum _PEP_rating {
    PEP_rating_undefined = 0,
    PEP_rating_cannot_decrypt,
    PEP_rating_have_no_key,
    PEP_rating_unencrypted,
    PEP_rating_unencrypted_for_some,
    PEP_rating_unreliable,
    PEP_rating_reliable,
    PEP_rating_trusted,
    PEP_rating_trusted_and_anonymized,
    PEP_rating_fully_anonymous,

    PEP_rating_mistrust = -1,
    PEP_rating_b0rken = -2,
    PEP_rating_under_attack = -3
} PEP_rating;
```

Appendix B. Document Changelog

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

o draft-birk-peg-rating-00:

- * Initial version

Appendix C. Open Issues

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

- o Better explain usage of Color Codes in Per-Identity Privacy Rating
- o Decide whether rating code scalars 6 and 7-9 should be raised to leave space for future extensions
- o Add Security Considerations
- o Add more source code excerpts to Appendix

- o Add rating codes for secure cryptographic methods and parameters and reference them

Authors' Addresses

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

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

Bernie Hoeneisen
Ucom Standards Track Solutions GmbH
CH-8046 Zuerich
Switzerland

Phone: +41 44 500 52 44
Email: bernie@ietf.hoeneisen.ch ([bernhard.hoeneisen AT ucom.ch](mailto:bernhard.hoeneisen@ucom.ch))
URI: <https://ucom.ch/>

httpbis
Internet-Draft
Intended status: Standards Track
Expires: December 27, 2018

B. Schwartz
Google
June 25, 2018

Hybrid Encapsulation Layer for IP and UDP Messages (HELIUM)
draft-schwartz-httpbis-helium-00

Abstract

HELIUM is a protocol that can be used to implement a UDP proxy, a VPN, or a hybrid of these. It is intended to run over a reliable, secure substrate transport. It can serve a variety of use cases, but its initial purpose is to enable HTTP proxies to forward non-TCP flows.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 27, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Overview	2
2. HELIUM Inner Protocol (HIP)	3
2.1. Terminology	3
2.2. Requirements	4
2.3. Abstract Structure	4
2.3.1. Error codes	6
2.4. CBOR-based Encoding (HIP-CBOR)	7
2.5. Addressing	8
2.5.1. IP Header	9
2.5.2. UDP Header	9
2.6. Example Configurations	9
2.6.1. Single IP tunnel	9
2.6.2. Multiple source IPs in one context	9
2.6.3. Domain-based proxy	10
2.6.4. UDP proxy with PMTUD and traceroute	10
2.6.5. Advanced DNS queries	10
2.6.6. UDP Server Application	11
2.6.7. High-Performance Delay-based Congestion Control	11
2.7. Optimizations	11
3. WebSocket as a HELIUM Substrate (HELIUM-WebSocket)	12
3.1. Direct Configuration	12
3.2. Implicit Configuration from an HTTP proxy	12
3.3. Optimizations	13
4. IANA Considerations	13
5. Acknowledgements	13
6. References	13
6.1. Normative References	13
6.2. Informative References	14
Author's Address	15

1. Overview

This proposal describes a network tunnel that is intended as a natural extension or complement to existing HTTP proxies. It has two components

- o A flexible packet-oriented tunneling protocol that can act as either a VPN or a UDP proxy (Section 2)
- o A substrate for this protocol that allows it to run as part of an HTTPS server (Section 3)

This design combines the benefits of several existing protocols, such as [OpenConnect] and [TURN]. Like OpenConnect, this protocol gains the privacy, authentication, and management benefits of HTTPS. Like

TURN, this protocol can be used as a UDP proxy for realtime and P2P applications.

2. HELIUM Inner Protocol (HIP)

The protocol is designed to span two different use cases

- o a UDP tunnel (proxy)
- o an IP tunnel (VPN)

These two use cases are normally handled by entirely separate protocols, like [TURN] and [L2TP]. However, UDP is fundamentally very similar to IP (differing mostly by the addition of a 2-byte "port number"), so it seems plausible that a single protocol may serve both purposes. Additionally, a UDP proxy can be enriched by partial support for ICMP (enabling [PMTUD], traceroute, etc.), so there may be configurations that benefit from blending these uses.

The protocol is intended to run between a client and a proxy, on a substrate that provides confidentiality, integrity, flow control, congestion control, and reliability (at least optionally). It should take advantage of substrates that support out-of-order delivery, but still function acceptably on strictly ordered transports.

2.1. Terminology

- o Proxy: the server implementing this protocol, acting as a UDP proxy or IP tunnel endpoint
- o Client: the endpoint that is implementing this protocol on the client side
- o Destination: a service that the client is trying to reach through the proxy
- o Context: the identity of the transport session used to transfer messages between a client and the proxy (e.g. one WebSocket)
- o Substrate: the transport protocol used to transfer these messages (e.g. WebSocket)
- o Flow: a sequence of related packets between the client and a single destination

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.

2.2. Requirements

- o It shall be possible for a proxy to operate in an environment without elevated privileges.
 - * Such a proxy might only support operating as a UDP tunnel.
- o It shall be possible for a proxy with elevated privileges to operate without any parsing of IP payloads.
 - * Such a proxy would operate as an IP tunnel.
- o A client can direct the proxy to send multiple packets from the same IP (and UDP port).
- o A client can tell what IP address and port the proxy is using to communicate on its behalf.
 - * A client can bind an address (or address:port) and learn it before emitting any packets.
- o A client can tell if the proxy doesn't support a feature it's trying to use.
- o New connections can be established without waiting for a roundtrip between client and proxy.
- o The protocol enables good performance when tunneling streams that use delay-based congestion control (e.g. TCP Vegas, [BBR], [RMCAT-GCC]).
- o The client has an option to let the proxy resolve DNS names itself, with a latency benefit.
- o The proxy can be implemented with tightly bounded memory usage.

2.3. Abstract Structure

Each HIP message consists of a type, optional metadata, and at most one packet (or prefix of a packet). The packet (or prefix) is a standard [IPv4] or [IPv6] packet, starting with the IP header.

There are three message types defined: "outbound", "inbound", and "meta".

A message from the client to the proxy is always of type "outbound". It always includes a complete packet for the proxy to send to the destination (potentially after header modifications). The possible metadata fields that this message may contain are as follows:

- o id (integer): An ID number identifying this message. If present, the client is implicitly requesting a "meta" message from the proxy. A client **MUST NOT** reuse an ID until a "meta" reply message is received.
- o domain (UTF-8 string): A DNS name to override the destination IP. The proxy will perform an A or AAAA lookup, depending on the IP version of the included packet. The proxy will buffer the packet until name lookup completes. The proxy **SHOULD** avoid creating duplicate outstanding DNS queries, and **SHOULD** cache the result to provide a consistent mapping.
- o dns (integer): The presence of this option indicates that the client wishes to direct the packet to one of the proxy's preferred DNS servers. Its value is an index into the proxy's list of preferred recursive resolvers for this IP version, modulo the length of the list. This option overrides the destination IP, and **MUST NOT** appear in a message with the "domain" option.

A message from the proxy to the client may be of type "inbound" or "meta". An "inbound" message contains a packet that the proxy received from the destination, unmodified, including the IP header. It contains one metadata field:

- o timestamp (integer): A timestamp in microseconds modulo 2^{32} , indicating when the proxy received this packet from the destination. The absolute base time is unspecified, as this is only used for computing time differences. If the proxy reassembled the packet from fragments, this timestamp is the time when reassembly completed.

A "meta" message is only sent by the proxy to a client after it receives an "outbound" message with an ID from the client. If the proxy modified the outbound packet in any way, the "meta" message **MUST** contain a prefix of the outbound packet as sent, including any parts that were modified. Changes might include the source IP, destination IP, TTL, DSCP priority, UDP source port, etc. If there was an error, the proxy **MAY** include a modified prefix that would not have encountered the error (e.g. by changing the protocol ID from an unsupported protocol (e.g. TCP) to a supported protocol (e.g. UDP)). The message contains the following metadata:

- o `id` (integer): This is the ID number of the "outbound" message to which this is a reply.
- o `error` (Array of integer): If present, these error codes indicate why the proxy could not send the packet contained in the "outbound" message to the destination.
- o `timestamp` (integer): The time when the outbound packet was sent from the proxy to the destination, in the same format used for "inbound" messages. If there was an error, this is the time that the error was detected.

If the proxy receives a message from the client of an unrecognized type, and the message has an "id" field, the server SHOULD reply with a "meta" message matching that ID and indicating an "Unsupported message type" error.

If the proxy receives a message from the client with unknown metadata fields, it SHOULD ignore the unknown fields and process the message as normal.

If the proxy receives an "outbound" message with an all-zero destination address and no address-overriding metadata, the proxy SHOULD rewrite the packet for transmission and establish any required address or port mappings, but not attempt to send the packet. If an ID number is present, the proxy SHOULD reply with a "meta" message indicating success unless a non-address-related error occurred.

All messages can also include padding. Padding can be represented as a metadata field named "padding" whose value is discarded by the recipient.

All integer values defined in this section are non-negative. All metadata keys defined here MUST NOT appear more than once. Recipients SHOULD treat negative numbers and repeated keys as metadata parsing errors.

2.3.1. Error codes

These are the numeric error codes that the proxy may include in a "meta" message

Code	Error
1	Unsupported message type
2	Metadata parsing error
3	Unsupported IP version
4	Invalid IP header
5	Can't send fragment
6	Packet too large
7	Unsupported IP option
8	Unsupported protocol
9	No route to host
10	Network unreachable
11	Destination IP not allowed
12	Destination DNS name not allowed
13	DNS name has no address (NXDOMAIN)
14	DNS name resolution failed
15	General server failure
16	Usage limit exceeded

Additional error codes may be defined in the future.

2.4. CBOR-based Encoding (HIP-CBOR)

To encode abstract HIP messages into concrete form, we use a [CBOR]-based encoding. Other equivalent but incompatible encodings might be defined in the future.

In this encoding, each message is formed by concatenating a one-byte type field, the metadata encoded in CBOR, and the packet or packet-prefix.

Byte	Type
0x01	outbound
0x02	inbound
0x03	meta

Metadata is encoded in CBOR as a Map. For compactness, keys are integer-valued, with the following significance:

Key	Field
0	padding
1	id
2	domain
3	dns
4	timestamp
5	error

Additional message types and metadata fields may be defined in the future.

When sending a message, endpoints SHOULD use the most compact available encoding of each metadata value. When receiving a message, recipients are NOT REQUIRED to accept extremely inefficient or obscure encodings that are allowed by CBOR (e.g. Bignums, Decimal Fractions).

2.5. Addressing

There are two major modes of operation that a proxy might use: IP tunnel and UDP tunnel. Both operation modes require the proxy to inspect and possibly modify the IP header of the packet contained in an "outbound" message before sending the packet to the destination. The UDP tunnel mode in addition requires the proxy to inspect and possibly modify the UDP header in the IP payload.

2.5.1. IP Header

Initially, the client does not know the IP address that the proxy will use as the source IP for packets it sends to the destination. The protocol does not require the client to correctly populate the source IP in its outbound packets to the proxy. Rather, the client chooses any IP address, and the proxy will rewrite this address into one of its own outbound IP addresses. Within a single context, the proxy **MUST** maintain a stable address mapping with a reasonable lifetime, similar to Network Address Translation [NAT].

In IP tunnel mode, the proxy **MUST NOT** map multiple contexts to the same outbound IP address at the same time, as it would then be impossible to determine unambiguously where to direct packets received from the destination. These outbound IP addresses **MAY** be publicly routable, or they **MAY** be in a reserved range (e.g. [RFC1918], [RFC4193]), using [NAT] to reach the public internet.

2.5.2. UDP Header

In UDP tunnel mode, the proxy **MAY** also rewrite the UDP source port of a packet before sending it to the destination. The client has no way to initially know what source port the proxy will use in this mode, so the protocol does not require the client to correctly populate the source port in its outbound packets to the proxy. In UDP tunnel mode, the proxy **MAY** map the same outbound IP address to multiple contexts with overlapping lifetimes, but the proxy **SHOULD** ensure that each UDP port is only mapped to a single context (i.e. an endpoint-independent mapping policy as described in [RFC4787]). A proxy **MAY** violate this condition only if it serves a limited use case in which the correct context for an inbound packet will never be ambiguous.

2.6. Example Configurations

2.6.1. Single IP tunnel

The client sends outbound IP packets to the server with empty metadata, and with various destinations and protocols (e.g. ICMP, TCP, UDP). The proxy rewrites the source address of all packets to match the reserved IP address for this client, and forwards all incoming packets to the client.

2.6.2. Multiple source IPs in one context

A client sends IP packets to the proxy with various source addresses, and includes an ID number in each one. For each ID number, the server's "meta" reply reveals the proxy source IP that was mapped to the client's chosen source IP. Once the client has learned the

mapping, the client stops including an ID number in subsequent messages.

2.6.3. Domain-based proxy

The client sends its initial flight of packets with an ID number and a domain in the metadata, and all zeroes in the destination IP address. The "meta" replies indicate the rewritten destination IP address, which is the resolved location of the destination. The client then emits subsequent packets with this destination IP address, and omits all metadata.

If the proxy does not know the exact IP header used (e.g. because it is using the network through a UDP socket API), it will synthesize an approximate IP header for the "meta" replies.

2.6.4. UDP proxy with PMTUD and traceroute

The client sends "outbound" UDP packets with the ID set and varying size or TTL. The proxy MUST NOT fragment unless the packet is IPv4 and the DONT-FRAGMENT bit is unset.

If the proxy could not send the packet because it was too large, it MUST reply with an error (Packet too large) and SHOULD include a rewritten header indicating the maximum size.

If the proxy fragmented the packet, it will reply with success and a prefix including the size of the first fragment.

If the proxy modified the outbound TTL, it will indicate this in the reply prefix.

If the proxy receives an ICMP response (e.g. Time Exceeded, Fragmentation Needed), it MAY forward it to the client. To support this use case, it MUST do so.

A proxy with this behavior can be implemented without elevated permissions on most common operating systems (see [I-D.martinsen-tram-stuntrace]).

2.6.5. Advanced DNS queries

The client sends an "outbound" UDP packet to port 53 with an ID number set, and a "dns" metadata value of 0. This packet is a DNS query, perhaps for a DNSKEY, TLSA, or TXT record.

The proxy overwrites the destination IP address with the IP of its first DNS server and sends the outbound packet. It also sends a

"meta" message to the client, containing the IP header with this destination address, as well as the modified source address and port.

The client is now waiting for an "inbound" message containing a reply from this DNS server to the modified source address and port. If no reply is received within some timeout, the client retries. This time, it sets a "dns" value of 1, indicating that the retry should use the proxy's second DNS server, if one exists.

2.6.6. UDP Server Application

The client sends an "outbound" UDP packet with an ID number set and all zeros in the destination IP. The "meta" reply includes the rewritten source IP and port, which is bound to this context. The client can now inform third parties to send data to this IP and port.

2.6.7. High-Performance Delay-based Congestion Control

The client is sending and receiving a flow that uses delay-based congestion control. Between client and proxy, this flow is transmitted according to the congestion control behaviors of the HELIUM substrate. From the proxy to the destination, congestion control is the responsibility of the client and destination.

To monitor delay on the proxy-destination path, the client can include an ID number in every outbound message. This will cause the proxy to reply with a "meta" message, including the send timestamp. By comparing these send timestamps with the receive timestamps in inbound messages, the client can accurately monitor the round-trip time between proxy and destination.

If the proxy-destination roundtrip time is gradually increasing, the client can reduce its send rate below the limit imposed by the HELIUM substrate.

2.7. Optimizations

Proxies are NOT REQUIRED to perform reassembly of inbound IP fragments. Proxies MAY reassemble IP fragments, or they MAY forward each fragment independently to the client. This helps to limit proxy memory usage.

When the client sends an "outbound" message with the "domain" metadata, the proxy has to buffer the corresponding packet until the domain name is resolved. To limit memory usage, the proxy can "peek" at the query without removing it from the transport's receive buffer. The transport's flow control will then limit the amount of memory that the client can consume.

3. WebSocket as a HELIUM Substrate (HELIUM-WebSocket)

The HELIUM Inner Protocol (Section 2) requires a substrate transport to deliver messages between client and proxy. The WebSocket protocol is a suitable substrate. Each HIP-CBOR message (Section 2.4) can be sent as a WebSocket message of type "binary".

If a browser is configured to act as a HELIUM client, communicating with the proxy over a WebSocket, the WebSocket is controlled and terminated by the browser itself, not associated with any particular origin or webpage.

3.1. Direct Configuration

The location of a WebSocket HELIUM proxy is defined by a WebSocket URL, e.g. "wss://proxy.example/example-path". If the client knows the address of a WebSocket HELIUM proxy, then the client may simply connect to the proxy by establishing a WebSocket connection. The client's WebSocket handshake request MUST contain the "Sec-WebSocket-Protocol" header with value "helium-cbor" as well as an authorization header (e.g. Proxy-Authorization) if needed.

3.2. Implicit Configuration from an HTTP proxy

Operators that run both an HTTP proxy, defined by some http or https URL, as well as a WebSocket HELIUM proxy, SHOULD return a response containing a new header, "Helium-Proxy-URL", when a client sends a proxy-specific request (e.g. HTTP CONNECT) to the operator's HTTP proxy. This new header, containing the WebSocket address of the HELIUM proxy, allows clients to discover the existence and location of a HELIUM proxy when they already know about an associated HTTP proxy. Clients can then connect to the discovered HELIUM proxy as described above.

In cases where user-facing proxy configuration options are limited (e.g. a web browser's settings menu), a user may not be able to directly configure a HELIUM proxy even if they know its address. If an option for configuring a HTTP(S) proxy is available, however, the Helium-Proxy-URL header will allow a user to implicitly configure a WebSocket HELIUM proxy by entering an associated HTTP(S) proxy address.

A client with access to both an HTTP(S) proxy and a HELIUM proxy SHOULD use the HTTP(S) proxy for all connections that it can support, and use the HELIUM proxy for all other network activity.

3.3. Optimizations

After initiating the WebSocket connection, a client MAY send its initial HIP messages without waiting for the server's reply. This saves 1 RTT, similar to TLS False Start [FALSESTART].

Clients and proxies MAY negotiate WebSocket DEFLATE compression with context takeover (see Section 7 of [RFC7692]). This will replace consistent headers with back-references to the previous matching packet. On typical streams, this removes most of the IP and HIP-CBOR overhead, and can even compress the payload if it contains patterns that appear in each packet. However, implementers should use caution when combining compression and padding, as compression can render some padding schemes ineffective.

4. IANA Considerations

The names and numbers of the HIP message types, metadata fields, and error codes will each require a new IANA registry. Additionally, HELIUM-WebSocket will require registration of a new WebSocket Protocol ("helium-cbor") and a new HTTP header ("Helium-Proxy-URL").

5. Acknowledgements

Many thanks to Katharine Daly and Lucas Pardue for their early and extensive review of this proposal.

6. References

6.1. Normative References

- [CBOR] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [IPv4] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<https://www.rfc-editor.org/info/rfc791>>.
- [IPv6] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [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>>.

- [RFC7692] Yoshino, T., "Compression Extensions for WebSocket", RFC 7692, DOI 10.17487/RFC7692, December 2015, <<https://www.rfc-editor.org/info/rfc7692>>.
- [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>>.

6.2. Informative References

- [BBR] Cardwell, N., Cheng, Y., Yeganeh, S., and V. Jacobson, "BBR Congestion Control", draft-cardwell-iccrb-bbr-congestion-control-00 (work in progress), July 2017.
- [FALSESTART] Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", RFC 7918, DOI 10.17487/RFC7918, August 2016, <<https://www.rfc-editor.org/info/rfc7918>>.
- [I-D.martinsen-tram-stuntrace] Martinsen, P. and D. Wing, "STUN Traceroute", draft-martinsen-tram-stuntrace-01 (work in progress), June 2015.
- [L2TP] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, DOI 10.17487/RFC2661, August 1999, <<https://www.rfc-editor.org/info/rfc2661>>.
- [NAT] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <<https://www.rfc-editor.org/info/rfc3022>>.
- [OpenConnect] Mavrogiannopoulos, N., "The OpenConnect VPN Protocol Version 1.0", draft-mavrogiannopoulos-openconnect-00 (work in progress), September 2016.
- [PMTUD] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <<https://www.rfc-editor.org/info/rfc1191>>.
- [RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <<https://www.rfc-editor.org/info/rfc1918>>.

- [RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <<https://www.rfc-editor.org/info/rfc4193>>.
- [RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <<https://www.rfc-editor.org/info/rfc4787>>.
- [RMCAT-GCC] Holmer, S., Lundin, H., Carlucci, G., Cicco, L., and S. Mascolo, "A Google Congestion Control Algorithm for Real-Time Communication", draft-ietf-rmcat-gcc-02 (work in progress), July 2016.
- [TURN] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, DOI 10.17487/RFC5766, April 2010, <<https://www.rfc-editor.org/info/rfc5766>>.

Author's Address

Ben Schwartz
Google

Email: bemasc@google.com