Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2020

B. Hoeneisen
Ucom.ch
H. Marques
K. Bristol
pEp Foundation

October 31, 2019

pretty Easy privacy (pEp): Key Synchronization Protocol (KeySync) draft-hoeneisen-pep-keysync-01

Abstract

This document describes the pEp KeySync protocol, which is designed to perform secure peer-to-peer synchronization of private keys across devices belonging to the same user.

Modern users of messaging systems typically have multiple devices for communicating, and attempting to use encryption on all of these devices often leads to situations where messages cannot be decrypted on a given device due to missing private key data. Current approaches to resolve key synchronicity issues are cumbersome and potentially unsecure. The pEp KeySync protocol is designed to facilitate this personal key synchronization in a user-friendly manner.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2020.

Copyright Notice

Table of Contents

1. Introduction

The pretty Easy privacy (pEp) [I-D.birk-pep] protocols describe a set of conventions for the automation of operations traditionally seen as barriers to the use and deployment of secure end-to-end interpersonal messaging. These include, but are not limited to, key management, key discovery, and private key handling.

This document specifies the pEp KeySync protocol, a means for secure, decentralized, peer-to-peer synchronization of private keys across devices belonging to the same user, allowing that user to send and receive encrypted communications from any of their devices.

For pEp implementations, pEp KeySync is a critical part of the broader pEp Sync protocol, which is designed to be extensible to allow for the synchronization of additional user data, such as configuration settings and peer trust status information across a user's devices.

This document will provide a general description of pEp KeySync, including idealized use cases, diagrams, and examples of messages that may be generated during the KeySync process.

1.1. 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].

1.2. Terms

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

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

  Note: In pEp KeySync, the Handshake is used to authenticate own devices (the user normally compares the Trustwords directly by looking at the screens of the devices involved).

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

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

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

- Identity: The combination of a unique user identifier plus a specific address (email, network ID, URI, etc.). A single user may have multiple identities. See also [RFC4949].

- Device Group: All of a user's devices which have successfully completed the KeySync process, and are now configured to share user data, such as cryptographic keys, trust information, calendars, configurations, and other data as a result of that process. This data is synchronized through a common channel for a given identity. For example, an identity might be a specific email address, and the common channel would be a mailbox for that email address.

- Sole Device: A device which is not part of a Device Group.

- Grouped Device: A device which is already part of a Device Group.

- Beacon (message): A technical text message that is broadcast by Sole Devices and transmitted through a message sent to the channel of an identity. Other Sole Devices, or a Grouped Device of the same unique identity and using that identity's channel, can interpret this Beacon in order to initiate negotiation for the formation of a Device Group.

- Transaction ID (TID): A UUID version 4, variant 1 number generated by each device during the pEp KeySync process in order to identify the respective devices involved.

- Default Key: A key which is actually used for a given identity.

- Own Key: A Default Key for an own identity.

1.3. Problem Statement

Secure and private digital communication is becoming a necessity for many people. Encryption protocols which utilize key pairs are the most popular and easily implemented methods to ensure a message is authentic and can be trusted.

However, most modern users have multiple devices for communicating, and attempting to use encryption on all of these devices often leads to situations where messages cannot be decrypted on a given device due to missing private key data. For example, Alice sends an encrypted message to Bob, using the public key of a key pair that Bob generated on his laptop. When Bob attempts to decrypt the message on his mobile phone, the private key that he generated on his laptop is not available. As a result, Bob must either use his laptop to decrypt the message, or attempt to copy the correct private key to his mobile device, which may expose his private key to potential leaks or theft.

1.4. Main Challenge

The main challenge that pEp KeySync is designed to overcome is to perform the synchronization in a secure manner so that private keys are not leaked or exposed to theft.

Note: The case of an adversary getting physical access to the device itself is beyond the scope of this document.

1.5. Approach

The basic approach to solving the multiple-device decryption problem is to synchronize private keys among the devices of a user in a secure manner. pEp achieves this by having a user form a Device Group among their devices. During this process, a Handshake process occurs, and the user will be presented with a Trustwords dialog between any two devices at a time for pairing purposes. (cf. [I-D.birk-pep-trustwords]) Simply put, a user MUST manually complete the Trustwords dialog before the automatic and security-sensitive transfer of private key information can occur.

2. General Description

The pEp KeySync protocol allows a user to securely synchronize private key data for multiple identities across their various devices.

This synchronization process is decentralized and performed as a two-phase commit protocol structure (2PC). This structure ensures consensus among the devices at all stages of the KeySync process.

KeySync's 2PC transaction is accomplished through the implementation of a Finite-State Machine (FSM) on each pEp-enabled device. This FSM not only sends and receives network traffic, which allows devices to communicate with each other throughout the KeySync process, but also interacts with the pEp engine itself.

Once activated, the pEp KeySync protocol initiates the formation of a Device Group, and the user is guided through a Handshake process on their respective devices. A user can choose to reject or cancel this process at any time, from either device, and private key data is not exchanged until the group formation process is verified on both devices.

Once a Device Group is formed, a user can add additional devices to their group through the same joining procedure. Upon adding the new device to the existing Device Group, key data is synchronized among all Grouped Devices, allowing a user to communicate privately from any of their secure identities.

2.1. Use Cases for pEp KeySync

This section describes ideal-condition use cases for pEp KeySync. The focus is on the core procedures and on the scenarios where everything works. Unexpected user behavior, error handling, race conditions, etc., are generally omitted from this section in order to focus on the general concepts of pEp KeySync. Additional use cases will be discussed in further detail throughout Appendix A.

2.1.1. Form Device Group

Our user, Alice, has two devices that are configured with pEp-implementing messaging clients and share the same identity for her preferred communication channel. In our example, this is a communication channel with an email address. Let us call these devices Alice_Mobile and Alice_Tablet. Each device already has its own key pair, which was automatically generated by the pEp protocol. Neither device knows anything about the other.

Alice wants full communication capability from both of her devices, but currently cannot do so, as the devices do not know about each other. Alice will use pEp KeySync to form a Device Group and add her devices to it. This allows for the exchange of private key data among its devices, allowing Alice to have full communication capability on both of her devices.

2.1.2. Add New Device to Existing Device Group

Sometime after devices Alice_Mobile and Alice_Tablet have formed a Device Group (cf. Section 2.1.1), Alice buys another device, Alice_Laptop, which is also configured with pEp-implementing messaging clients and shares the same identity for her preferred communication channel (the aforementioned email address). Alice_Laptop also has a key pair, which was automatically generated by the pEp protocol, just as the Grouped Devices Alice_Mobile and Alice_Tablet have. But while the Grouped Devices know each other and have exchanged private keys, Alice_Laptop and the Grouped Devices don't know each other. Thus, Alice does not have full communication capability across the three devices.

As before with devices Alice_Mobile and Alice_Tablet, Alice will use pEp KeySync to add device Alice_Laptop to the existing Device Group, allowing all three devices to exchange private key information, and Alice to have access to her messages from any of them.

2.1.3. Exchange Private Keys

All devices from Alice are part of a Device Group (cf. Section 2.1.1 and Section 2.1.2). However, as keys may expire or get reset, it is inevitable that new key pairs will be generated. For Alice to maintain her ability to read all encrypted messages on all devices, any new private key needs to be shared with the other devices in the device group. All devices in Alice's Device Group will share the latest private keys as they are generated, keeping all of her devices up to date and functioning as desired.

2.1.4. Leave Device Group

Alice decides that her mobile phone, Alice_Mobile, should no longer have access to all private keys of the Device Group. Alice can manually tell her mobile phone to leave the Device Group by turning off the pEp Sync feature on her device, which deactivates KeySync. The Device Group is dissolved, and Sync is disabled on her mobile phone. This action also initiates the pEp KeyReset protocol, which resets keys for all own identities.

In the future, if Alice desires, she can re-add her mobile phone to a Device Group, but she will first have to re-enable Sync, and then initiate the joining procedure again (cf. Section 2.1.1 and Section 2.1.2).

2.1.5. Remove other Device from Device Group

One of Alice's devices may be stolen or become otherwise compromised. She needs to ensure that the affected device no longer receives updates to private keys from the other devices in her Device Group. Using one of her remaining Grouped Devices, Alice can disable pEp Sync (and thus KeySync) on her remaining devices. This action dissolves the Device Group and initiates the pEp KeyReset protocol.

2.2. Interaction Diagrams

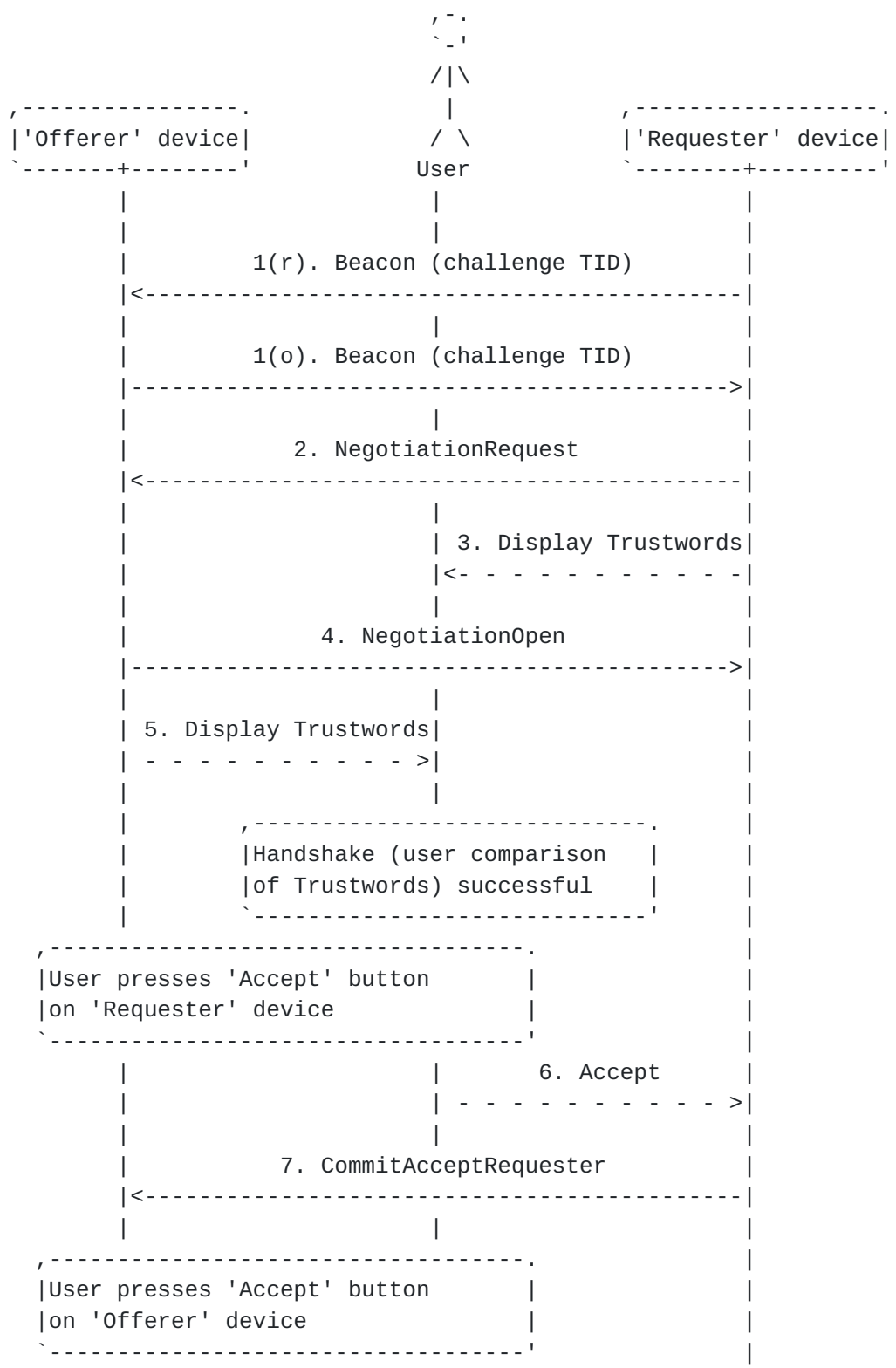
The following interaction diagrams depict what happens during Alice's KeySync scenarios in a simplified manner. For each scenario, we first present a successful case, then an unsuccessful case and, finally, a case that has been interrupted, or discontinued. Some details are skipped here for the sake of readability. Descriptions of the interactions are included after each diagram.

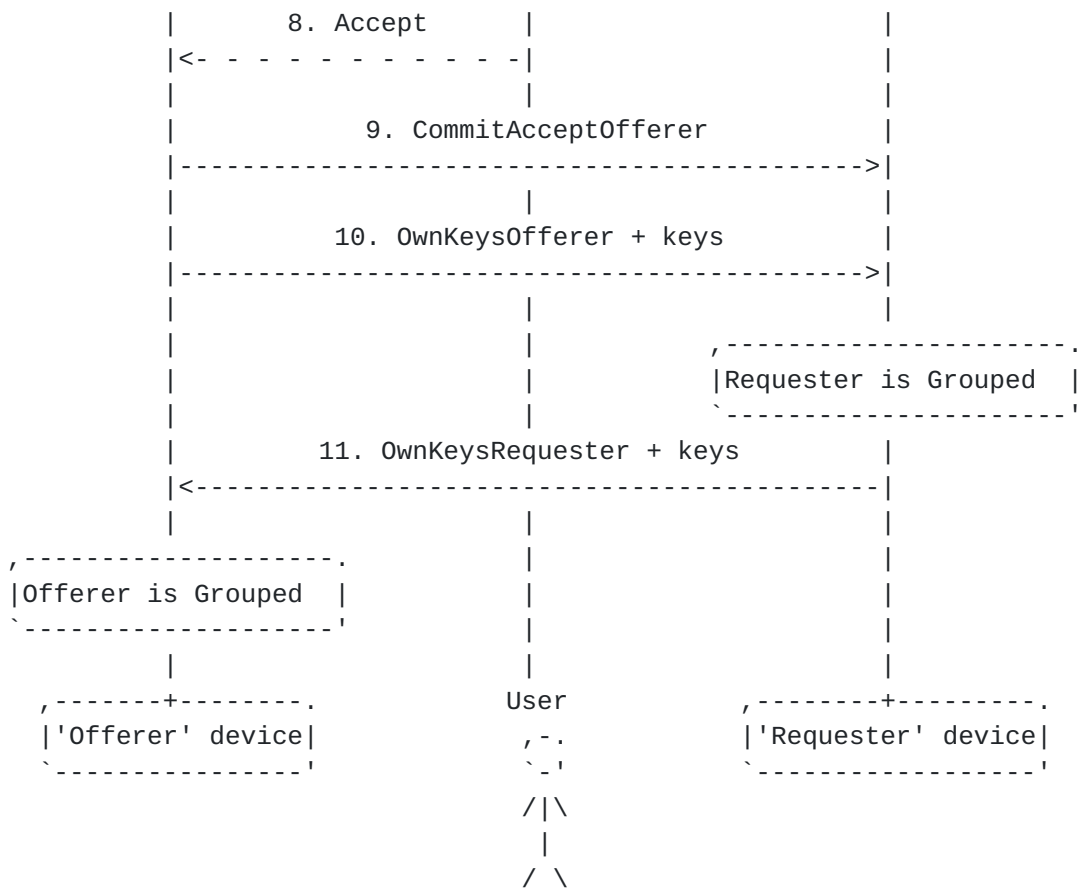
Each pEp-enabled device runs its own Finite-State Machine (FSM), which interact with each other throughout the KeySync process, and drive the UI options presented to the user. All messages are 'broadcast' between devices. The TIDs added to each message allow the identification of received messages which pertain to the ongoing transaction and its sender.

For events requiring user interaction in order to proceed, it does not matter which device has the specified option chosen first unless otherwise indicated. For example, if an event states that the 'Offerer' chooses 'Accept' to continue, the process will be unaffected if the 'Requester' device does so first. The only difference is that the order of the roles for the remainder of the given scenario will be swapped.

2.2.1. Form Device Group

2.2.1.1. Successful Case




8. Accept
9. CommitAcceptOfferer
10. OwnKeysOfferer + keys
Requester is Grouped
11. OwnKeysRequester + keys
Offerer is Grouped
'Offerer' device
User
'Requester' device

As depicted above, a user intends to form a Device Group in order to securely share key material among its members. The group is formed by an 'Offerer' device and a 'Requester' device. The names 'Offerer' and 'Requester' are derived from the FSM, in which the device roles are defined during the start sequence, which is necessary for the FSM to work as intended.

During initialization of pEp KeySync, each device generates a Transaction-ID (TID). These TIDs are sent as a challenge in a Beacon over the mutual channel, and the device roles of 'Offerer' and 'Requester' are determined by the numeric value of each device's unique TID.

1. Every device sends a Beacon message containing a challenge TID. Upon receipt of a Beacon message from another device, the received challenge TID is compared with the device's own challenge TID. The device which has a TID with a lower numerical value is assigned as the 'Requester', and the other device is automatically assigned as the 'Offerer'.

   Note: The 'Offerer' device MUST NOT start a negotiation. In the event the earlier Beacon message is lost, the 'Offerer' device re-sends its own Beacon and waits for a response. Message 1(r) depicts the Beacon message sent by the 'Requester' device and is not required for the process to continue.

2. After determination of the role, the 'Requester' device sends a NegotiationRequest message.

3. The 'Requester' device displays the Trustwords to the user.

4. Upon receipt of the NegotiationRequest message, the 'Offerer' device sends a NegotiationOpen message.

5. The 'Offerer' device displays the Trustwords to the user.

6. The user compares the Trustwords of both devices. As the Trustwords are the same on both devices, the user chooses the 'Accept' option on the 'Requester' device.

7. On receipt of the user's 'Accept', the 'Requester' device sends a CommitAcceptRequester message.

   The 'Offerer' device receives this message and waits for the user to choose 'Accept'.

8. The user compares the Trustwords of both devices and chooses the 'Accept' option on the 'Offerer' device.

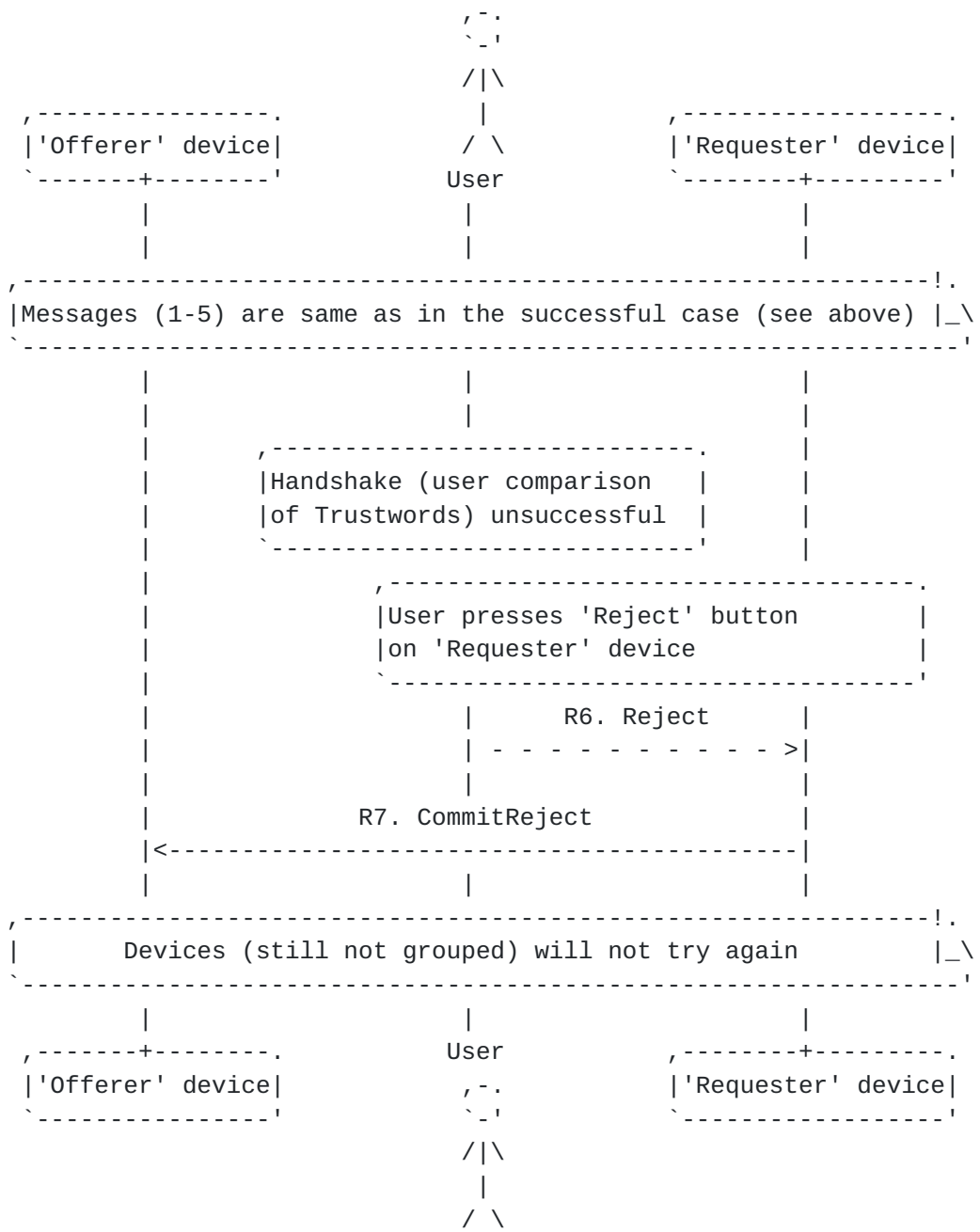
9. Once the user chooses 'Accept', the 'Offerer' device sends a CommitAcceptRequester message.

10. The 'Offerer' device then sends an OwnKeysOfferer message along with the user's local key pairs (private and public keys) to to be synchronized.

11. Upon receipt of the OwnKeysOfferer message, the 'Requester' device is grouped and sends an OwnKeysRequester message along with the user's local key pairs (private and public keys) to be synchronized.

    Upon receipt of the OwnKeysRequester message, the 'Offerer' device is also grouped. The formation of the Device Group has been successful.

2.2.1.2. Unsuccessful Case



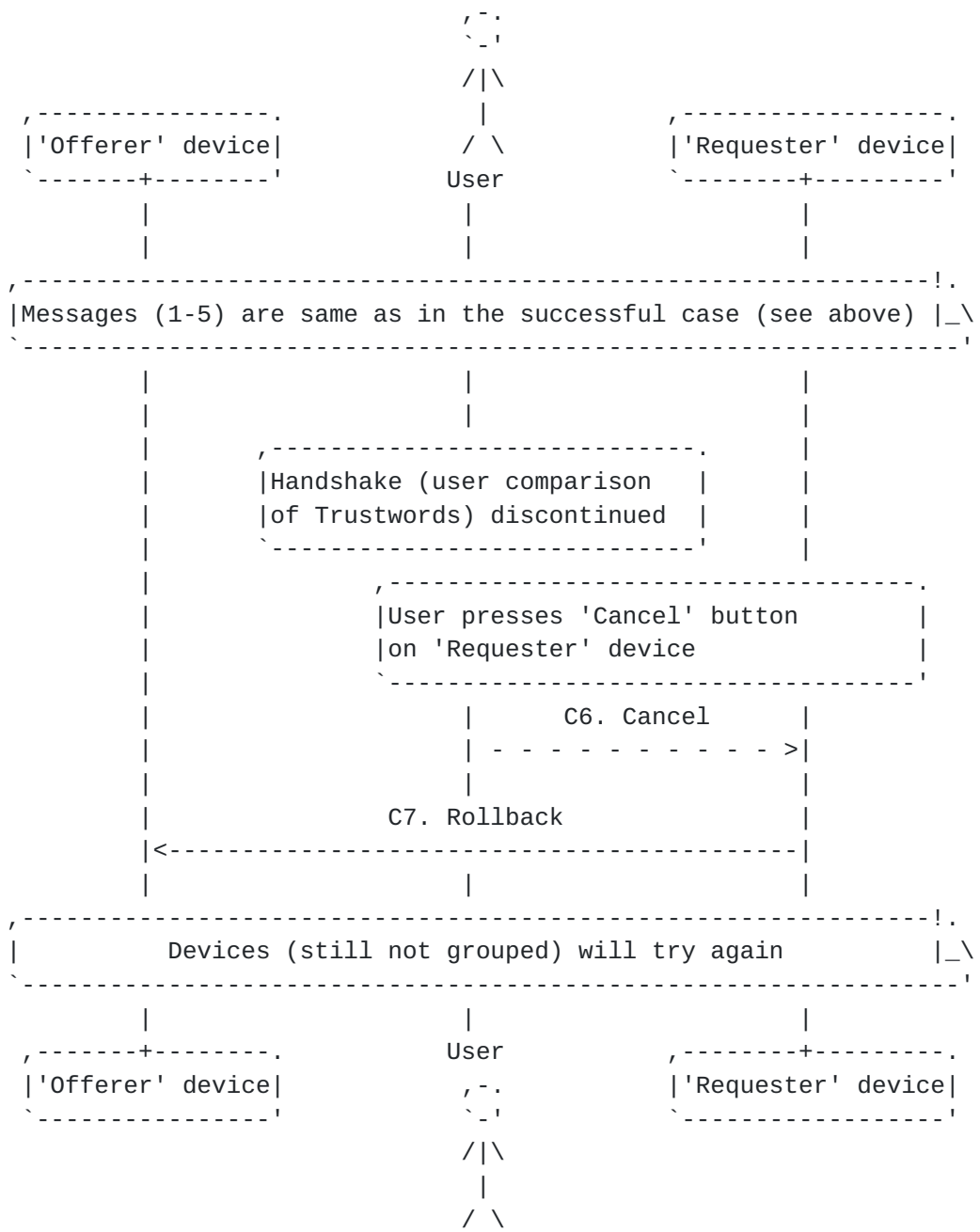
For unsuccessful KeySync attempts, messages 1-5 are the same as in a successful attempt (see above), but once the Trustwords are shown, events are as follows:

R6. The user compares the Trustwords of both devices. As the Trustwords do not match, the user chooses the 'Reject' option on the 'Requester' device.

R7. On receipt of the user's 'Reject', the 'Requester' device sends a CommitReject message.

Once the CommitReject message is sent or received, respectively, the devices cannot form a Device Group, and pEp KeySync is disabled on both devices. As a result, there are no further attempts to form a Device Group involving either of these two devices. KeySync may be re-enabled in the pEp settings on the affected device(s).

2.2.1.3. Discontinuation Case



For discontinued (canceled) KeySync attempts, messages 1-5 are the same as in a successful attempt (see above), but once the Trustwords are shown, events are as follows:

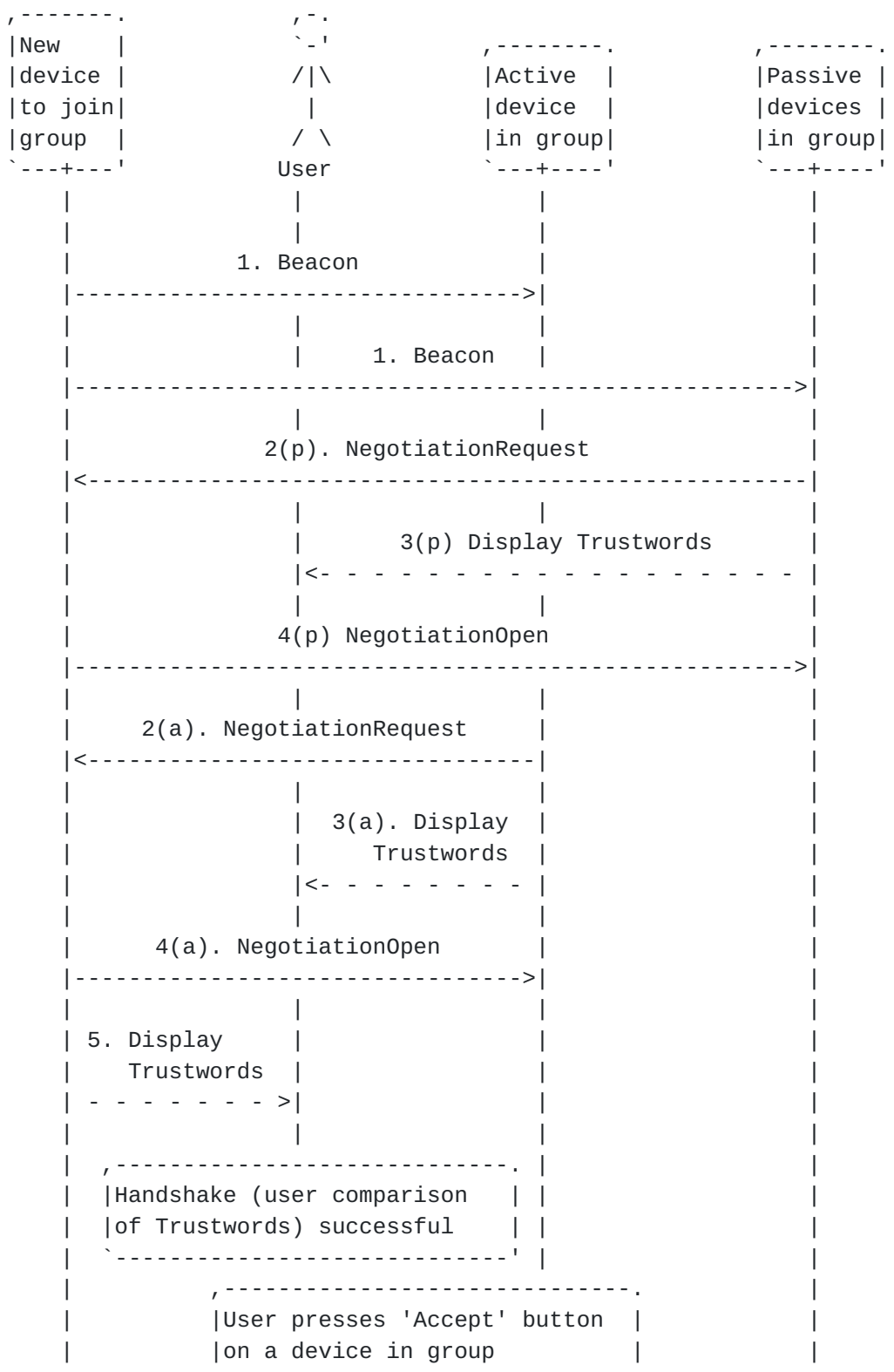
C6. The user decides to discontinue the process and chooses the 'Cancel' option on the 'Requester' device.

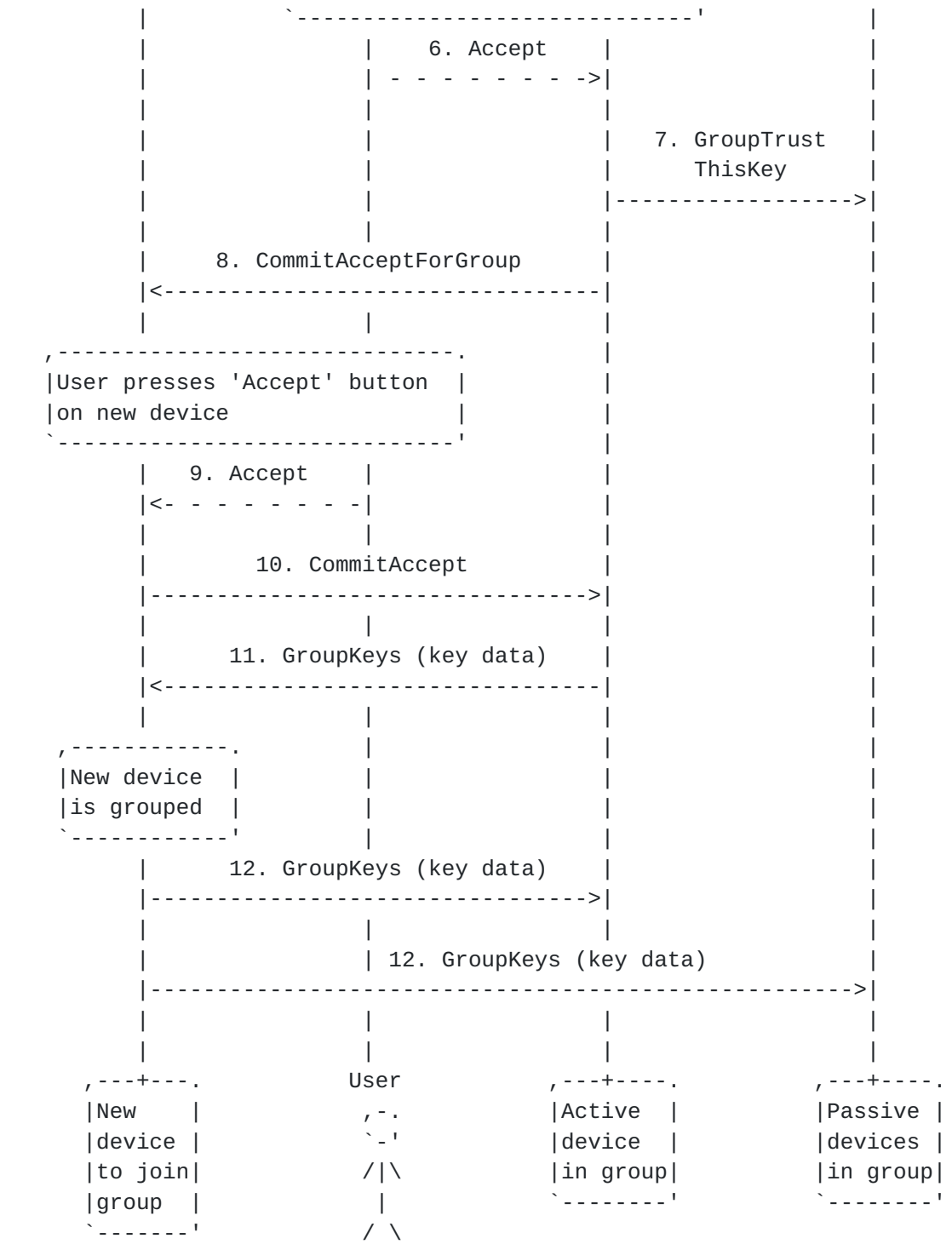
C7. On receipt of the user's 'Cancel', the 'Requester' device sends a rollback message.

The devices do not form a Device Group. KeySync remains enabled on both devices, and forming a Device Group can start again.

2.2.2. Add New Device to Existing Device Group

2.2.2.1. Successful Case




6. Accept
7. GroupTrust ThisKey
8. CommitAcceptForGroup
User presses 'Accept' button on new device
9. Accept
10. CommitAccept
11. GroupKeys (key data)
New device is grouped
12. GroupKeys (key data)
12. GroupKeys (key data)
User
New device to join group
Active device in group
Passive devices in group

As depicted above, a user intends to add a new device to an existing Device Group.

1. During initialization of pEp KeySync, the new device sends a Beacon message.

   Note: In the diagram, all messages marked "1. Beacon" are a single message, but drawn separately in order to convey that the message is sent to all devices in the Device Group.

2. Upon receipt of a Beacon message from a device not part of a Device Group, all Grouped Devices send a NegotiationRequest message.

   Note: Messages 2(a) and 2(p) are different instances of the NegotiationRequest message type.

3. All Grouped Devices display the Trustwords to the user.

4. Upon receipt of every NegotiationRequest message, the New Device sends a NegotiationOpen message.

   Note: Messages 4(a) and 4(p) are different instances of the NegotiationOpen message type.

5. The new device displays the Trustwords to the user.

6. The user compares the Trustwords of both devices and chooses the 'Accept' option on one of the Grouped Devices.

   Note 1: The Grouped Device that the user chooses the 'Accept' option from assumes the role of the active device for the Device Group.

7. On receipt of the user's 'Accept', the Active Grouped Device sends a TrustThisKey message to the passive Grouped Devices.

8. The Active Grouped Device also sends a CommitAcceptForGroup message to the new device. Upon receipt, the new device waits for the user to choose 'Accept'.

9. The user compares the Trustwords of both devices and chooses the 'Accept' option on the new device.

10. Once the user chooses 'Accept', the new device sends a CommitAccept message to the Device Group.

11. The new device then sends a GroupKeys message which contains its own private keys.

12. Upon receipt of the GroupKeys message from the new device, the Active Grouped Device FSM transitions to state Grouped, adds the new device's keys to the GroupKeys, and sends a GroupKeys message to the entire Device Group. The new device has successfully joined the Device Group and all keys are synchronized among the devices.

Note: In the diagram, all messages marked "12. GroupKeys + keys" are a single message, but drawn separately in order to convey that the message is sent to all devices in the Device Group.

2.2.2.2. Unsuccessful Case

For unsuccessful KeySync attempts, messages 1-5 are the same as in a successful attempt (see above), but once the Trustwords are shown, events are as follows:

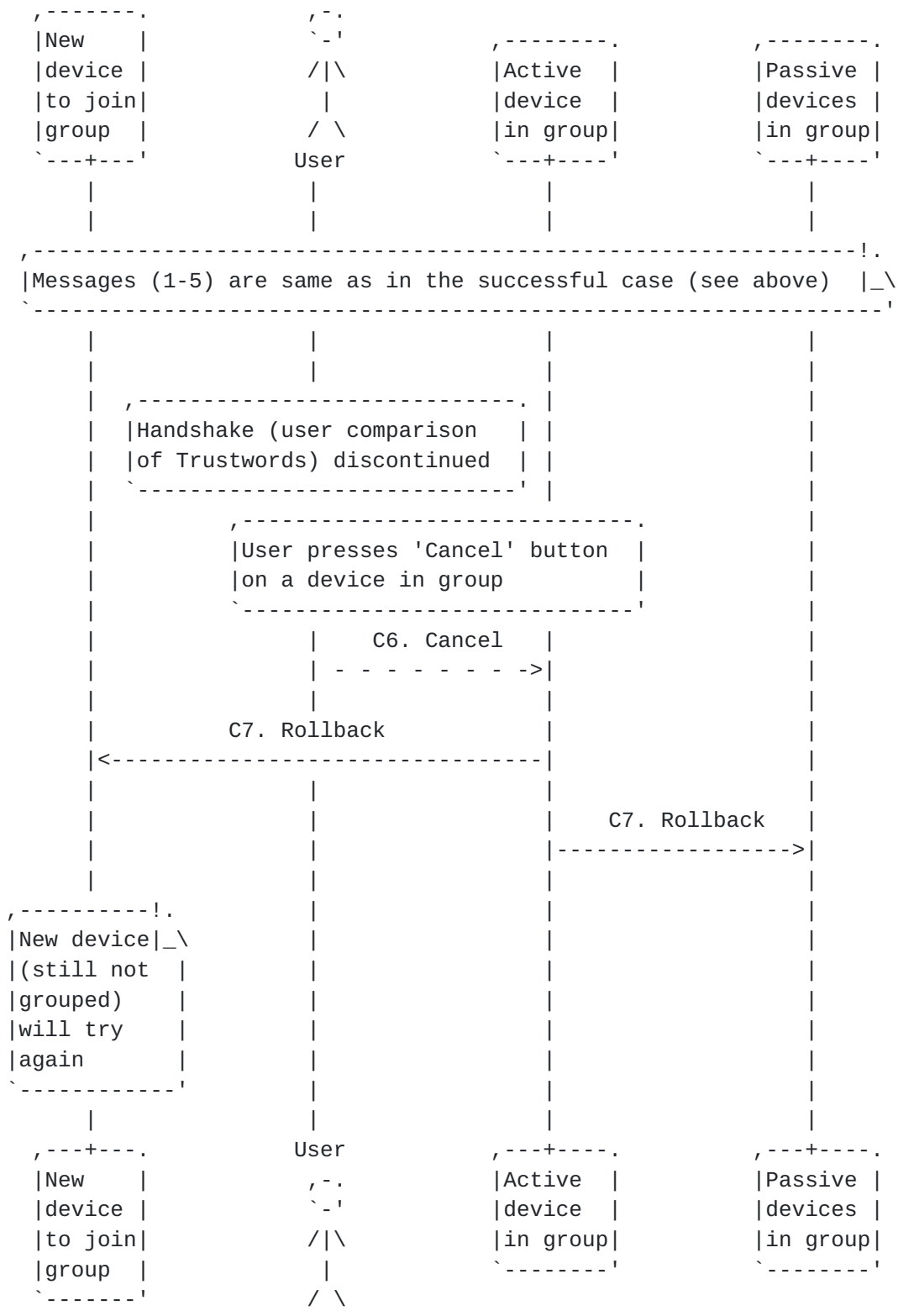
R6. The user compares the Trustwords displayed on both devices. If the Trustwords do not match, the user chooses the 'Reject' option on one of the Grouped Devices. This issues a 'CommitReject' message to the FSM as well as all devices in the Device Group.

R7. Upon receiving the 'Reject' message from the Device Group, the 'Requester' device sends a 'CommitReject' message to the FSM.

Note: In the diagram, all messages marked "R7. CommitReject" are a single message, but drawn separately in order to convey that the message is sent to all devices in the Device Group.

Once the CommitReject message is sent or received, respectively, the new device cannot join the Device Group, and pEp KeySync is disabled on the new device. As a result, there are no further attempts to join a Device Group by the new device. pEp KeySync may be re-enabled in the pEp settings on the affected device.

2.2.2.3. Discontinuation Case



For discontinued (canceled) KeySync attempts, messages 1-5 are the same as in a successful attempt (see above), but once the Trustwords are shown, events are as follows:

C6. The user decides to discontinue the process and chooses the 'Cancel' option on one of the Grouped Devices.

C7. On receipt of the 'Cancel' from the 'Offerer' device, the 'Requester' device sends a rollback message to the FSM.

Note: In the diagram, all messages marked "C7. Rollback" are a single message, but drawn separately in order to convey that the message is sent to all devices in the Device Group.

The new device does not join the Device Group. KeySync remains enabled and joining a Device Group can start again at any time.

2.2.3. Exchange Private Keys

[[TODO]]

2.2.4. Leave Device Group

[[TODO]]

2.2.5. Remove other Device from Device Group

[[TODO]]

3. Security Considerations

[[TODO]]

4. Privacy Considerations

[[TODO]]

5. IANA Considerations

This document has no actions for IANA.

6. Acknowledgments

The authors would like to thank the following people who have provided significant contributions to actual Running Code and the development of this document: Volker Birk and Krista Bennett.

Furthermore, the authors would like to thank the following people who provided helpful comments and suggestions for this document: Claudio Luck, Damian Rutz, Damiano Boppart, and Nana Karlstetter.

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

[I-D.birk-pep] Marques, H., Luck, C., and B. Hoeneisen, "pretty Easy privacy (pEp): Privacy by Default", draft-birk-pep-04 (work in progress), July 2019.

[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>.

7.2. Informative References

[I-D.birk-pep-trustwords] Hoeneisen, B. and H. Marques, "IANA Registration of Trustword Lists: Guide, Template and IANA Considerations", draft-birk-pep-trustwords-04 (work in progress), July 2019.

[I-D.marques-pep-handshake] Marques, H. and B. Hoeneisen, "pretty Easy privacy (pEp): Contact and Channel Authentication through Handshake", draft-marques-pep-handshake-03 (work in progress), July 2019.

[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/>.

Appendix A. Reference Implementation

[[Note: The full Finite-State Machine code can be found in Appendix B. This section is not a complete reference at this time. The authors intend to refine this section in future revisions of this document.]]

The pEp KeySync Finite-State Machine is based on a two-phase commit protocol (2PC) structure. This section describes the states, actions, events, and messages which comprise the pEp KeySync FSM, and are intended to allow readers to understand the general functionality and message flow of the FSM.

States are used to direct actions, events, and messages. Actions describe internal FSM functions, and fall into two general types. The first action type directs the state transitions within the FSM, and the second type drives UI functionality. Events are exchanged both between negotiation partners as well as the pEp engine itself to trigger actions and send messages. Messages contain information to ensure the integrity of the KeySync session as well as additional data, depending on the type of message (cf. Appendix A.2).

A.1. Full Finite-State Machine Diagram

A full diagram of the implemented pEp KeySync FSM can be found at the following URL:

https://pep.foundation/dev/repos/internet-drafts/raw-file/tip/misc/figures/sync/sync_fsm_full.svg

A.1.1. States

A.1.1.1. InitState

On initialization, the FSM enters InitState, which evaluates and determines a device's group status. If the device is detected to belong to a Device Group, the FSM transitions to state Grouped. Otherwise, the FSM transitions to state Sole (cf. Appendix A.1.2.1).

A.1.1.2. Sole

This is the default FSM state for an ungrouped device.

On initialization, a challenge TID is created and sent out inside of a Beacon message along with the device's current state. The FSM also listens for Beacons from other devices. Upon receipt of a Beacon message from another device, the received challenge TID is compared with the own challenge. The device with the lower challenge TID is assigned the 'Requester' role, and the other device is automatically assigned the 'Offerer' role.

If a device is determined to be the 'Requester', it issues a NegotiationRequest event to the 'Offerer'.

When the 'Offerer' device receives this NegotiationRequest message, it responds with a NegotiationOpen message, and the 'Offerer' FSM transitions to state HandshakingOfferer while it awaits the 'Requester' device response.

On receipt of the 'Offerer' device's NegotiationOpen message, the 'Requester' FSM proceeds in one of two ways, depending on the 'Offerer' device state:

- Sole: The 'Requester' FSM transitions to state HandshakingRequester.

- Grouped: The 'Requester' FSM transitions to state HandshakingToJoin.

A.1.1.3. HandshakingOfferer

This state can only be entered by the 'Offerer' device in a Sole state, and drives user interface options, including the Trustwords dialog. The user is prompted to compare Trustwords and choose from the following options:

- Accept: A CommitAcceptOfferer message is sent to the 'Requester' device, and the FSM transitions to state HandshakingPhase1Offerer while it waits for a response.

- Reject: A CommitReject message is sent to the 'Requester' device, pEp KeySync is disabled, and the FSM transitions to state End.

- Cancel: A Rollback message is sent to the 'Requester' device, and the FSM transitions to state Sole.

Once the user selects one of the above options on the 'Offerer' device, the FSM waits for a response from the 'Requester' device. When this response is received, the 'Offerer' FSM performs a sameNegotiationAndPartner conditional check to verify the session integrity. If this conditional returns 'true', the FSM proceeds as follows, depending on the message received:

- CommitAcceptRequester: The 'Requester' device public key is trusted, and the FSM transitions to state HandshakingPhase2Offerer.
- CommitReject: pEp KeySync is disabled, and the FSM transitions to state End.
- Rollback: The FSM transitions to state Sole.

A.1.1.4. HandshakingRequester

This state can only be entered by the 'Requester' device in a Sole state, and drives user interface options, including the Trustwords dialog. The user is prompted to compare Trustwords, and choose from the following options:

- Accept: A CommitAcceptRequester message is sent to the 'Offerer' device, the 'Offerer' public key is trusted, and the FSM transitions to state HandshakingPhase1Requester while it waits for a response.
- Reject: A CommitReject message is sent to the 'Offerer' device, pEp KeySync is disabled, and the FSM transitions to state End.
- Cancel: A Rollback message is sent to the 'Offerer' device, and the FSM transitions to state Sole.

Once the user selects one of the above options on the 'Requester' device, the FSM waits for a response from the 'Offerer' device. When this response is received, the 'Requester' FSM performs a sameNegotiationAndPartner conditional check to verify the session integrity. If this conditional returns 'true', the FSM proceeds as follows, depending on the message received:

- CommitAcceptOfferer: The FSM transitions to state HandshakingPhase2Requester.
- CommitReject: pEp KeySync is disabled, and the FSM transitions to state End.
- Rollback: The FSM transitions to state Sole.

A.1.1.5. HandshakingPhase1Offerer

This state can only be entered by the 'Offerer' device in a Sole state.

This state awaits and processes the response from a 'Requester' device in state HandshakingRequester. When this response is received, the 'Offerer' FSM performs a sameNegotiationAndPartner conditional check to verify the session integrity. If this conditional returns 'true', the FSM proceeds as follows, depending on the message received:

- CommitAcceptRequester: The FSM transitions to state FormingGroupOfferer.
- CommitReject: The 'Requester' public key is mistrusted, pEp KeySync is disabled, and the FSM transitions to state End.
- Rollback: The 'Requester' public key is mistrusted, and the FSM transitions to state Sole.

A.1.1.6. HandshakingPhase1Requester

This state can only be entered by the 'Requester' device in a Sole state.

This state awaits and processes the response from an 'Offerer' device in state HandshakingOfferer. When this response is received, the 'Requester' FSM performs a sameNegotiationAndPartner conditional check to verify the session integrity. If this conditional returns 'true', the FSM proceeds as follows, depending on the message received:

- CommitAcceptOfferer: The FSM transitions to state FormingGroupRequester.
- CommitReject: The 'Offerer' public key is mistrusted, pEp KeySync is disabled, and the FSM transitions to state End.
- Rollback: The 'Offerer' public key is mistrusted, and the FSM transitions to state Sole.

A.1.1.7. HandshakingPhase2Offerer

This state can only be entered by the 'Offerer' device in a Sole state.

The FSM waits for the 'Offerer' device in state HandshakingPhase1Offerer to process the 'Requester' device response. Once that is done, the following UI options are displayed for the 'Offerer' device:

- Accept: The 'Requester' public key used in the Handshake is marked 'trusted', a CommitAcceptOfferer message is issued to the 'Requester', and the FSM transitions to state FormingGroupOfferer.

- Reject: A CommitReject message is issued to the 'Requester' device, pEp KeySync is disabled, and the FSM transitions to state End.

- Cancel: A Rollback message is issued to the 'Requester' device, and the FSM transitions to state Sole.

A.1.1.8. HandshakingPhase2Requester

This state can only be entered by the 'Requester' device in a Sole state.

The FSM waits for the 'Requester' device in state HandshakingPhase1Requester to process the 'Offerer' device response. Once that is done, the following UI options are displayed for the 'Requester' device:

- Accept: The 'Offerer' public key used in the Handshake is marked 'trusted', a CommitAcceptRequester message is issued to the 'Offerer' device, and the FSM transitions to state FormingGroupRequester.

- Reject: A CommitReject message is issued to the 'Offerer' device, pEp KeySync is disabled, and the FSM transitions to state End.

- Cancel: A Rollback message is issued to the 'Offerer' device, and the FSM transitions to state Sole.

A.1.1.9. FormingGroupOfferer

This state can only be entered by the 'Offerer' device in a Sole state.

In this state, the user is given two options: Initialize the final step of the KeySync process (exchange of private key information), or Cancel.

- Init: The FSM prepares the Own Keys on the 'Offerer' device for synchronization. The FSM then issues an OwnKeysOfferer message

which contains these keys, triggers a UI event to indicate that Device Group formation is in progress, and awaits a response from the 'Requester' device.

Once the 'Requester' responds with a OwnKeysRequester message, the 'Requester' device keys are received, combined with the 'Offerer' keys, and saved in a shared GroupKeys array (saveGroupKeys). The 'Requester' device keys are marked as default for those respective identities (receivedKeysAreDefaultKeys). A UI event (showGroupCreated) indicates that the Device Group process is complete, and the FSM transitions to state Grouped.

- Cancel: A Rollback message is issued to the 'Requester' device, and the FSM transitions to state Sole. No private key data is exchanged.

A.1.1.10. FormingGroupRequester

This state can only be entered by the 'Requester' device in a Sole state.

In this state, the user is given two options: Initialize the final step of the KeySync process (exchange of private key information), or Cancel.

- Init: The FSM triggers a UI event to indicate that Device Group formation is in progress, and awaits an OwnKeysOfferer response from the 'Offerer' device.

- Cancel: A Rollback message is issued to the 'Requester' device, and the FSM transitions to state Sole. No private key data is exchanged.

Once an OwnKeysOfferer message is received, the 'Requester' FSM saves the 'Offerer' keys in a shared GroupKeys array (saveGroupKeys), and prepares the device's Own Keys for synchronization. The 'Requester' device keys are marked as default for those respective identities (ownKeysAreDefaultKeys). The FSM then issues an OwnKeysRequester message which contains these keys. A UI event (showGroupCreated) indicates that the Device Group process is complete, and the FSM transitions to state Grouped.

A.1.1.11. Grouped

This is the default state for any Grouped Device.

On initialization, this state generates a new challenge TID and shows the device as being in the Grouped state. This device state also

listens for Beacons from other devices that are not yet part of the Device Group.

Upon receipt of a Beacon message from Sole Device, the device sends a NegotiationRequest message and waits for the Sole Device to respond with a NegotiationOpen message.

On receipt of the NegotiationOpen message from the Sole Device, the FSM of the Grouped Device stores the negotiation information and transitions to state HandshakingGrouped.

In this state, other events may also be processed, but these events do not result in a transition to another state.

A.1.1.12. HandshakingToJoin

This state can only be entered by a device in the Sole state that is attempting to join an existing Device Group.

In this state, the FSM waits for the user to compare the Trustwords and to choose from the following options on the new device:

- Accept: A CommitAccept message is sent and the FSM transitions to state HandshakingToJoinPhase1.

- Reject: A CommitReject message is sent, pEp KeySync is disabled (on the new device), and the FSM transitions to state End.

- Cancel: A Rollback message is sent and the FSM transitions to state Sole.

If the 'Accept' option is chosen on a Grouped Device, a CommitAcceptForGroup message is received and the FSM transitions to state HandshakingToJoinPhase2.

If the 'Reject' option is chosen on a Grouped Device, a CommitReject message is received. pEp KeySync is disabled (on the new device) and the FSM transitions to state End.

If the 'Cancel' option is chosen on the 'Requester' device, a Rollback message is received and the FSM transitions to state Sole.

A.1.1.13. HandshakingToJoinPhase1

This state is entered by a new device only, i.e. a device that is not yet part of a Device Group. The FSM waits for the user to finish the Handshake on a Grouped Device (the user compares the Trustwords and chooses from the options presented):

If the 'Accept' option is chosen on a Grouped Device, a CommitAcceptForGroup message is received and the FSM transitions to state JoiningGroup.

If the 'Reject' option is chosen on a Grouped Device, a CommitReject message is received. pEp KeySync is disabled (on the new device) and the FSM transitions to state End.

If the 'Cancel' option is chosen on the 'Requester' device, a Rollback message is received and the FSM transitions to state Sole.

A.1.1.14. HandshakingToJoinPhase2

This state is entered by a new device only, i.e. a device that is not yet part of a Device Group.

In this state, the FSM waits for the user to compare the Trustwords and to choose from the following options on the new device:

- Accept: A CommitAccept message is sent and the FSM transitions to state JoiningGroup.
- Reject: A CommitReject message is sent, pEp KeySync is disabled (on the new device), and the FSM transitions to state End.
- Cancel: A Rollback message is sent and the FSM transitions to state Sole.

A.1.1.15. JoiningGroup

This state is entered by a new device only, i.e. a device that is not yet part of a Device Group.

The FSM waits to receive the keys from the active Grouped Device. Once received, these are saved and marked as Default Keys. Then it sends all keys to the Grouped Devices and the FSM transitions to state Grouped.

A.1.1.16. HandshakingGrouped

This state is entered by Grouped Devices only, i.e., devices that are part of a Device Group.

In this state the FSM waits for the user to compare the Trustwords and to choose any of the following options from any Grouped Device, which will now become the Active Grouped Device:

- Accept: A CommitAcceptForGroup message is sent and the FSM transitions to state HandshakingGroupedPhase1.

- Reject: A CommitReject message is sent and the FSM transitions to state Grouped.

- Cancel: A Rollback message is sent and the FSM transitions to state Grouped.

If the 'Accept' option is chosen on the new device, a CommitAccept message is received and the FSM transitions to state HandshakingPhase2.

If the 'Reject' option is chosen on the new device, a CommitReject message is received and the FSM transitions to state Grouped.

If the 'Cancel' option is chosen on the new device, a Rollback message is received and the FSM transitions to state Grouped.

Note: In this state, other events are processed, but these events do not result in a transition to another state and are not discussed here.

A.1.1.17. HandshakingGroupedPhase1

This state is entered by Grouped Devices only, i.e., devices that are already part of a Device Group. The FSM waits for the user to finish the Handshake on the new device (the user compares the Trustwords and chooses from the options presented):

If the 'Accept' option is chosen on the new device, a CommitAccept message is received and the FSM transitions to state Grouped.

If the 'Reject' option is chosen on the new device, a CommitReject message is received and the FSM transitions to state Grouped.

If the 'Cancel' option is chosen on the new device, a Rollback message is received and the FSM transitions to state Grouped.

In this state also a various other events are processed, which do not result in a transition to another state.

A.1.1.18. HandshakingGroupedPhase2

This state is entered by Grouped Devices only, i.e. devices that are already part of a Device Group.

In this state the FSM waits for the user to compare the Trustwords and to choose from the following options from any Grouped Device, which will now become the Active Grouped Device:

- Accept: A CommitAcceptForGroup message is sent and the FSM transitions to state HandshakingGroupedPhase1
- Reject: A CommitReject message is sent and the FSM transitions to state Grouped
- Cancel: A Rollback message is sent and the FSM transitions to state Grouped

In this state also various other events are processed, which do not result in a transition to another state, but in the execution of certain actions (e.g., saveGroupKeys).

A.1.2. Actions

A.1.2.1. deviceGrouped (conditional)

The 'deviceGrouped' conditional evaluates true if a device is already in a Device Group. This boolean value is available and eventually altered locally on every KeySync-enabled device. For example, in the reference implementation, this boolean value is stored in a local SQL database.

The 'deviceGrouped' value is what the KeySync FSM uses upon initialization to determine whether a device should transition to state Sole or state Grouped.

A.1.2.2. disable

The 'disable' action may be called in an number of scenarios. For example, a user has rejected a pEp Handshake on either device involved in a pEp Handshake. At this time, in all cases, invoking the 'disable' function results in the FSM transitioning to state End, which disables the KeySync feature. pEp KeySync can be manually re-enabled in the pEp settings on an affected device.

A.1.2.3. hideHandshakeDialog

During the negotiation process for adding a new device to an existing Device Group, any device in that group can be used to complete the handshaking process and, as a result, each device in a Device Group will display the Handshake dialog options. Once this process is performed on one of the Grouped Devices, that device becomes the Active Grouped Device, and a GroupTrustThisKey message is sent to the

other (now Passive) Grouped Devices. Upon receipt of the GroupTrustThisKey message, the 'hideHandshakeDialog' action is invoked, and is intended to force the closure of the extra Handshake dialog boxes.

A.1.2.4. openNegotiation

An 'openNegotiation' action is carried out either by a Sole Device in the 'Requester' role, or a Grouped device upon receipt of a Beacon message from another Sole Device. Most importantly, this action ensures that the own TID and the challenge TID of the Sole Device get combined by the mathematical XOR function. In this way, a common TID exists which can be used by both devices a user wishes to pair. This TID is crucial in allowing the devices to recognize themselves in a particular pairing process, as multiple pairing process can occur simultaneously.

A.1.2.5. ownKeysAreDefaultKeys

The ownKeysAreDefaultKeys action is invoked by the 'Requester' device during the final step of Device Group formation between two Sole devices, and ensures that the Own Keys for the identities on the 'Requester' device are set as the default for those respective identities.

A.1.2.6. newChallengeAndNegotiationBase

The newChallengeAndNegotiationBase action is invoked by a device during an Init event in either the Sole or Grouped state, and serves to clear and generate a new challenge TID and negotiation state.

A.1.2.7. partnerIsGrouped (conditional)

The partnerIsGrouped conditional evaluates whether a negotiation partner is already in a Device Group or not, which determines if the new device will be joining the Device Group or forming a new Device Group with another device in the Sole state. If this boolean evaluates true, then the FSM transitions to HandshakingToJoin. If not, the FSM proceeds with the negotiation process for two Sole Devices seeking to form a new Device Group.

A.1.2.8. prepareOwnKeys

The prepareOwnKeys action is invoked during the latter phases of the KeySync protocol for both new and existing Device Group joining processes. This action indicates to a device that all key information that has been selected for synchronization should be prepared for sending to the other negotiation partner.

A.1.2.9. receivedKeysAreDefaultKeys

The receivedKeysAreDefaultKeys action marks the keys received during Device Group formation as the Default Keys for the respective identities of the device which sent them.

A.1.2.10. sameChallenge (conditional)

The sameChallenge action compares the challenge TIDs sent in Beacons. If this boolean evaluates 'true', the received challenge TID originated from that device, meaning that the received Beacon has come back to us and we can ignore it. If the boolean evaluates 'false', then the TID comparison takes place and the roles of 'Offerer' and 'Requester' are assigned. The lower of the two numbers is the 'Requester'.

A.1.2.11. sameNegotiation (conditional)

As with sameChallenge, the sameNegotiation action evaluates 'true' if the FSM finds a NegotiationRequest message that a Sole Device sent out is determined to be self-originating. The Transaction ID (TID) will be an exact match upon comparison, and the NegotiationRequest will be ignored as a result.

A.1.2.12. sameNegotiationAndPartner (conditional)

The sameNegotiationAndPartner action serves as a session fidelity check. If this boolean evaluates 'true', it confirms that the pEp KeySync session in progress is the same throughout, and that the negotiation partner has not changed.

A.1.2.13. saveGroupKeys

The saveGroupKeys action directs the addition of any keys received during a KeySync process to a GroupKeys array, along with any existing Own or Grouped Device Keys.

A.1.2.14. showBeingInGroup

A.1.2.15. showBeingSole

The 'showBeingSole' action in state Sole drives a UI event that can be used to notify a pEp user that their device is Sole (ungrouped), and guide the user through creation or joining of a Device Group. This action also prepares the necessary structures to potentially initiate KeySync, in the event a Beacon from another device is received.

A.1.2.16. showDeviceAdded

The 'showDeviceAdded' action drives a UI event that is used to notify a pEp user that a Sole Device was added to an already existing Device Group.

A.1.2.17. showGroupCreated

In either role that a Sole Device can assume ('Requester' or 'Offerer'), the action 'showGroupCreated' drives a UI event which notifies a user that a new Device Group was formed from two Sole Devices.

A.1.2.18. showGroupedHandshake

The 'showGroupedHandshake' action drives a UI event on a Grouped device, which a pEp implementer should use to display a pEp Handshake dialog. This dialog should indicate that there is a new Sole Device that is requesting to join the Device Group that this Grouped device belongs to.

A.1.2.19. showJoinGroupHandshake

The 'showJoinGroupHandshake' action drives a UI event on a Sole Device attempting to join an existing Device Group, and should be used by pEp implementers to show a Handshake dialog on the Sole Device.

A.1.2.20. showSoleHandshake

For cases where two Sole Devices are attempting to form a new Device Group, the showSoleHandshake action drives a UI event which a pEp implementer should use to display a pEp Handshake dialog to each of the devices in negotiation.

A.1.2.21. storeNegotiation

The storeNegotiation action saves the received non-own negotiation information, which is used by the sameNegotiationAndPartner action to perform a session fidelity check (cf. Appendix A.1.2.11).

A.1.2.22. tellWeAreGrouped

The tellWeAreGrouped action is used by devices already in the Grouped state. It is sent in a Beacon and indicates to Sole Devices that they are entering a negotiation with a Grouped device. For the Sole Device, receiving this action determines which state the FSM will transition to next.

A.1.2.23. tellWeAreNotGrouped

The 'tellWeAreNotGrouped' action is used by Sole Devices which are assigned the role of 'Requester' upon challenge TID comparison, and is sent along with a NegotiationRequest event to indicate to the 'Offerer' device that they are entering into a negotiation request with a Sole Device.

A.1.2.24. trustThisKey

The trustThisKey action is executed in all states when a user chooses 'Accept' on the Handshake dialog. Trust for the public key from the negotiation partner is granted so the rest of the KeySync process can be conducted securely. The trust also extends to the private key portion of the key pair at later stage in the KeySync process, so long as the user continues to choose 'Accept' on both devices. If the process is canceled or rejected at any point after the public key trust has been granted, that trust will be removed (cf. Appendix A.1.2.25).

A.1.2.25. untrustThisKey

If the 'Cancel' or 'Reject' options are chosen at any point during the KeySync process after a negotiation partner's public key has been trusted, trust on that public key is removed (cf. Appendix A.1.2.24). The untrustThisKey action ensures that the negotiation partner's public key can never be attached to messages sent to outside peers from the recipient device.

A.1.2.26. useOwnChallenge

Once a Beacon is received by a device in either the Sole or Grouped state, the useOwnChallenge action attaches the device's generated challenge TID to an outgoing Beacon or NegotiationRequest event for comparison and session verification purposes.

A.1.3. Transitions

Transitions are changes between states within the FSM, and are indicated by the 'go' command throughout the code. Please see the desired State for additional information on why and when these changes are triggered.

A.2. Messages

[[TODO]]

A.2.1. Format

[[TODO]]

A.2.2. List of Messages Used in Finite-State Machine

A.2.2.1. Beacon

[[TODO]]

Please find more information in the following code excerpt:

```
message Beacon 2, type=broadcast, security=unencrypted {
    field TID challenge;
    auto Version version;
}
```

A.2.2.2. NegotiationRequest

[[TODO]]

Please find more information in the following code excerpt:

```
message NegotiationRequest 3, security=untrusted {
    field TID challenge;
    auto Version version;
    field TID negotiation;
    field bool is_group;
}
```

A.2.2.3. NegotiationOpen

[[TODO]]

Please find more information in the following code excerpt:

```
message NegotiationOpen 4, security=untrusted {
    auto Version version;
    field TID negotiation;
}
```

A.2.2.4. Rollback

[[TODO]]

Please find more information in the following code excerpt:

```
message Rollback 5, security=untrusted {
    field TID negotiation;
}
```

A.2.2.5. CommitReject

[[TODO]]

Please find more information in the following code excerpt:

```
message CommitReject 6, security=untrusted {
    field TID negotiation;
}
```

A.2.2.6. CommitAcceptOfferer

[[TODO]]

Please find more information in the following code excerpt:

```
message CommitAcceptOfferer 7, security=untrusted {
    field TID negotiation;
}
```

A.2.2.7. CommitAcceptRequester

[[TODO]]

Please find more information in the following code excerpt:

```
message CommitAcceptRequester 8, security=untrusted {
    field TID negotiation;
}
```

A.2.2.8. CommitAccept

[[TODO]]

Please find more information in the following code excerpt:

```
message CommitAccept 9, security=untrusted {
    field TID negotiation;
}
```

A.2.2.9. CommitAcceptForGroup

[[TODO]]

Please find more information in the following code excerpt:

```
message CommitAcceptForGroup 10, security=untrusted {
    field TID negotiation;
}
```

A.2.2.10. GroupTrustThisKey

[[TODO]]

Please find more information in the following code excerpt:

```
// default: security=trusted only
message GroupTrustThisKey 11 {
    field Hash key;
}
```

A.2.2.11. GroupKeys

[[TODO]]

Please find more information in the following code excerpt:

```
// trust in future
message GroupKeys 12, security=attach_own_keys {
    field IdentityList ownIdentities;
}
```

A.2.2.12. OwnKeysOfferer

[[TODO]]

Please find more information in the following code excerpt:

```
        message OwnKeysOfferer 13, security=attach_own_keys {
            field IdentityList ownIdentities;
        }
```

A.2.2.13. OwnKeysRequester

[[TODO]]

Please find more information in the following code excerpt:

```
        message OwnKeysRequester 14, security=attach_own_keys {
            field IdentityList ownIdentities;
        }
```

A.2.3. Example Messages

[[TODO]]

Appendix B. Finite-State Machine Code

Below is the full code for the pEp KeySync FSM, including messages and external events.

```
fsm KeySync 1, threshold=300 {
    version 1, 2;

    state InitState {
        on Init {
            if deviceGrouped
                go Grouped;
            go Sole;
        }
    }

    state Sole timeout=off {
        on Init {
            do newChallengeAndNegotiationBase;
            do showBeingSole;
            send Beacon;
        }
```

```
on KeyGen {
    send Beacon;
}

on CannotDecrypt { // cry baby
    send Beacon;
}

on Beacon {
    if sameChallenge {
        // this is our own Beacon; ignore
    }
    else {
        if weAreOfferer {
            do useOwnChallenge;
            send Beacon;
        }
        else /* we are requester */ {
            do openNegotiation;
            do tellWeAreNotGrouped;
            // requester is sending NegotiationRequest
            send NegotiationRequest;
            do useOwnChallenge;
        }
    }
}

on NegotiationRequest {
    if sameChallenge { // challenge accepted
        if sameNegotiation {
            // this is our own NegotiationRequest; ignore
        }
        else {
            do storeNegotiation;
            // offerer is accepting by
            // confirming NegotiationOpen
            send NegotiationOpen;
            if partnerIsGrouped
                go HandshakingToJoin;
            else
                go HandshakingOfferer;
        }
    }
}

on NegotiationOpen if sameNegotiationAndPartner {
    // requester is receiving NegotiationOpen
    do storeNegotiation;
```

```
            go HandshakingRequester;
        }
    }

    // handshaking without existing Device group
    state HandshakingOfferer timeout=600 {
        on Init
            do showSoleHandshake;

        // Cancel is Rollback
        on Cancel {
            send Rollback;
            go Sole;
        }

        on Rollback if sameNegotiationAndPartner
            go Sole;

        // Reject is CommitReject
        on Reject {
            send CommitReject;
            do disable;
            go End;
        }

        on CommitReject if sameNegotiationAndPartner {
            do disable;
            go End;
        }

        // Accept means init Phase1Commit
        on Accept {
            do trustThisKey;
            send CommitAcceptOfferer;
            go HandshakingPhase1Offerer;
        }

        // got a CommitAccept from requester
        on CommitAcceptRequester if sameNegotiationAndPartner
            go HandshakingPhase2Offerer;
    }

    // handshaking without existing Device group
    state HandshakingRequester timeout=600 {
        on Init
            do showSoleHandshake;

        // Cancel is Rollback
```

```
            on Cancel {
                send Rollback;
                go Sole;
            }

            on Rollback if sameNegotiationAndPartner
                go Sole;

            // Reject is CommitReject
            on Reject {
                send CommitReject;
                do disable;
                go End;
            }

            on CommitReject if sameNegotiationAndPartner {
                do disable;
                go End;
            }

            // Accept means init Phase1Commit
            on Accept {
                do trustThisKey;
                send CommitAcceptRequester;
                go HandshakingPhase1Requester;
            }

            // got a CommitAccept from offerer
            on CommitAcceptOfferer if sameNegotiationAndPartner
                go HandshakingPhase2Requester;
        }

        state HandshakingPhase1Offerer {
            on Rollback if sameNegotiationAndPartner {
                do untrustThisKey;
                go Sole;
            }

            on CommitReject if sameNegotiationAndPartner {
                do untrustThisKey;
                do disable;
                go End;
            }

            on CommitAcceptRequester if sameNegotiationAndPartner {
                go FormingGroupOfferer;
            }
        }
```

```
        state HandshakingPhase1Requester {
            on Rollback if sameNegotiationAndPartner {
                do untrustThisKey;
                go Sole;
            }

            on CommitReject if sameNegotiationAndPartner {
                do untrustThisKey;
                do disable;
                go End;
            }

            on CommitAcceptOfferer if sameNegotiationAndPartner {
                go FormingGroupRequester;
            }
        }

        state HandshakingPhase2Offerer {
            on Cancel {
                send Rollback;
                go Sole;
            }

            on Reject {
                send CommitReject;
                do disable;
                go End;
            }

            on Accept {
                send CommitAcceptOfferer;
                do trustThisKey;
                go FormingGroupOfferer;
            }
        }

        state HandshakingPhase2Requester {
            on Cancel {
                send Rollback;
                go Sole;
            }

            on Reject {
                send CommitReject;
                do disable;
                go End;
            }
```

```
            on Accept {
                send CommitAcceptRequester;
                do trustThisKey;
                go FormingGroupRequester;
            }
        }

        state FormingGroupOfferer {
            on Init {
                do prepareOwnKeys;
                send OwnKeysOfferer; // not grouped yet,
                                     // this are our own keys
                do showFormingGroup;
            }

            on Cancel {
                send Rollback;
                go Sole;
            }

            on Rollback
                go Sole;

            on OwnKeysRequester {
                do saveGroupKeys;
                do receivedKeysAreDefaultKeys;
                do showGroupCreated;
                go Grouped;
            }
        }

        state FormingGroupRequester {
            on Init
                do showFormingGroup;

            on Cancel {
                send Rollback;
                go Sole;
            }

            on Rollback
                go Sole;

            on OwnKeysOfferer {
                do saveGroupKeys;
                do prepareOwnKeys;
                do ownKeysAreDefaultKeys;
                send OwnKeysRequester;
```

```
                do showGroupCreated;
                go Grouped;
            }
        }

        state Grouped timeout=off {
            on Init {
                do newChallengeAndNegotiationBase;
                do showBeingInGroup;
            }

            on GroupKeys
                do saveGroupKeys;

            on KeyGen {
                do prepareOwnKeys;
                send GroupKeys;
            }

            on Beacon {
                do openNegotiation;
                do tellWeAreGrouped;
                send NegotiationRequest;
                do useOwnChallenge;
            }

            on NegotiationOpen if sameNegotiationAndPartner {
                do storeNegotiation;
                go HandshakingGrouped;
            }

            on GroupTrustThisKey
                do trustThisKey;
        }

        // sole device handshaking with group
        state HandshakingToJoin {
            on Init
                do showJoinGroupHandshake;

            // Cancel is Rollback
            on Cancel {
                send Rollback;
                go Sole;
            }

            on Rollback if sameNegotiationAndPartner
                go Sole;
```

```
            // Reject is CommitReject
            on Reject {
                send CommitReject;
                do disable;
                go End;
            }

            on CommitAcceptForGroup if sameNegotiationAndPartner
                go HandshakingToJoinPhase2;

            on CommitReject if sameNegotiationAndPartner {
                do disable;
                go End;
            }

            // Accept is Phase1Commit
            on Accept {
                do trustThisKey;
                send CommitAccept;
                go HandshakingToJoinPhase1;
            }
        }

        state HandshakingToJoinPhase1 {
            on Rollback if sameNegotiationAndPartner
                go Sole;

            on CommitReject if sameNegotiationAndPartner {
                do disable;
                go End;
            }

            on CommitAcceptForGroup if sameNegotiationAndPartner
                go JoiningGroup;
        }

        state HandshakingToJoinPhase2 {
            on Cancel {
                send Rollback;
                go Sole;
            }

            on Reject {
                send CommitReject;
                do disable;
                go End;
            }
```

```
            on Accept {
                do trustThisKey;
                go JoiningGroup;
            }
        }

        state JoiningGroup {
            on GroupKeys {
                do saveGroupKeys;
                do receivedKeysAreDefaultKeys;
                do prepareOwnKeys;
                send GroupKeys;
                do showDeviceAdded;
                go Grouped;
            }
        }

        state HandshakingGrouped {
            on Init
                do showGroupedHandshake;

            // Cancel is Rollback
            on Cancel {
                send Rollback;
                go Grouped;
            }

            on Rollback if sameNegotiationAndPartner
                go Grouped;

            // Reject is CommitReject
            on Reject {
                send CommitReject;
                go Grouped;
            }

            on CommitReject if sameNegotiationAndPartner
                go Grouped;

            // Accept is Phase1Commit
            on Accept {
                do trustThisKey;
                send GroupTrustThisKey;
                send CommitAcceptForGroup;
                go HandshakingGroupedPhase1;
            }

            on CommitAccept if sameNegotiationAndPartner
```

```
            go HandshakingGroupedPhase2;

        on GroupTrustThisKey {
            do hideHandshakeDialog;
            do trustThisKey;
        }

        on GroupKeys
            do saveGroupKeys;
    }

    state HandshakingGroupedPhase1 {
        on Rollback if sameNegotiationAndPartner
            go Grouped;

        on CommitReject if sameNegotiationAndPartner
            go Grouped;

        on CommitAccept if sameNegotiationAndPartner {
            do prepareOwnKeys;
            send GroupKeys;
            go Grouped;
        }

        on GroupTrustThisKey {
            do trustThisKey;
        }

        on GroupKeys
            do saveGroupKeys;
    }

    state HandshakingGroupedPhase2 {
        on Cancel {
            send Rollback;
            go Grouped;
        }

        on Reject {
            send CommitReject;
            go Grouped;
        }

        on Accept {
            do trustThisKey;
            send GroupTrustThisKey;
            do prepareOwnKeys;
            send GroupKeys;
```

```
            go Grouped;
        }

        on GroupTrustThisKey {
            do trustThisKey;
        }

        on GroupKeys
            do saveGroupKeys;
    }

    external Accept 129;
    external Reject 130;
    external Cancel 131;

    // beacons are always broadcasted

    message Beacon 2, type=broadcast, security=unencrypted {
        field TID challenge;
        auto Version version;
    }

    message NegotiationRequest 3, security=untrusted {
        field TID challenge;
        auto Version version;
        field TID negotiation;
        field bool is_group;
    }

    message NegotiationOpen 4, security=untrusted {
        auto Version version;
        field TID negotiation;
    }

    message Rollback 5, security=untrusted {
        field TID negotiation;
    }

    message CommitReject 6, security=untrusted {
        field TID negotiation;
    }

    message CommitAcceptOfferer 7, security=untrusted {
        field TID negotiation;
    }

    message CommitAcceptRequester 8, security=untrusted {
        field TID negotiation;
```

```
        }

        message CommitAccept 9, security=untrusted {
            field TID negotiation;
        }

        message CommitAcceptForGroup 10, security=untrusted {
            field TID negotiation;
        }

        // default: security=trusted only
        message GroupTrustThisKey 11 {
            field Hash key;
        }

        // trust in future
        message GroupKeys 12, security=attach_own_keys {
            field IdentityList ownIdentities;
        }

        message OwnKeysOfferer 13, security=attach_own_keys {
            field IdentityList ownIdentities;
        }

        message OwnKeysRequester 14, security=attach_own_keys {
            field IdentityList ownIdentities;
        }
    }
```

Appendix C. Document Changelog

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

- draft-hoeneisen-pep-KeySync-00:
  - Initial version
- draft-hoeneisen-pep-KeySync-01:
  - Major rewrite of upper sections
  - Adjust to reflect code changes
  - Move Finite-State Machine reference and code to Appendices A & B

Appendix D. Open Issues

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

o Resolve several TODOs / add missing text

LocalWords: Boppart boolean showFormingGroup broadcasted

Authors' Addresses

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

Phone: +41 44 500 52 40
Email: bernie@ietf.hoeneisen.ch (bernhard.hoeneisen AT ucom.ch)
URI: https://ucom.ch/

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

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

Kelly Bristol
pEp Foundation
Oberer Graben 4
CH-8400 Winterthur
Switzerland

Email: kelly@pep-project.org
URI: https://pep.foundation/