SecureJoin: Protecting chat messaging against network adversaries

Release 0.20.0

Delta Chat implementors

Dec 18, 2023

Contents

SecureJoin protocols provide a usable model for message end-to-end encryption which is secure against attackers trying to break authenticity, confidentiality or integrity of messages as can occur with compromised servers and networks. They are implemented, user-tested and continuously refined in production-releases of the cross-platform Delta Chat messenger[1]. Other messenger implementors as well as researchers are welcome to submit remarks, questions or critique either through github[2] or by contacting Delta Chat teams[3].

Today's SecureJoin protocols are derived from respective key parts of CounterMitm protocols[4] which were produced by researchers of the NEXTLEAP EU project (2016-2018) to explore decentralized messaging and privacy topics.

[1] https://delta.chat
[2] https://github.com/deltachat/securejoin
[3] https://delta.chat/en/contribute
[4] https://countermitm.readthedocs.io/en/latest/

1 Introduction

This document documents and discusses SecureJoin protocols as implemented by the Delta Chat messenger[5]. In particular, this document considers how to secure Autocrypt[6]-capable mail apps against active network attackers. Autocrypt aims to achieve convenient end-to-end encryption of e-mail. The Level 1 Autocrypt specification offers users opt-in e-mail encryption, but only considers passive adversaries. Active network adversaries, who could, for example, tamper with the Autocrypt header during e-mail message transport, are not considered in the Level 1 specification. Yet, such active attackers might undermine the security of Autocrypt. Therefore, we present and discuss SecureJoin protocols as a new, practically usable way to prevent and detect active network attacks against Autocrypt[7]-capable mail apps.

This document reflects the current state of implementation (Delta Chat app version 1.42) and may evolve in the future to address user feedback, implementation and security considerations. To design SecureJoin protocols, we consider usability, cryptographic and implementation aspects simultaneously, because they constrain and complement each other. Note that the basis for SecureJoin protocols was laid in the first two sections of CounterMitm[8] which was created by researchers of NEXTLEAP, a 2016-2018 project on privacy and decentralized messaging, funded through the EU Horizon 2020 programme.

1.1 Attack model and terminology

We consider

- *Peer* devices that use network messages for transporting messages and for key-exchange in order to establish end-to-end encryption.
- A *network adversary* that can read, modify, and create network messages. Examples of such an adversary are an ISP, an e-mail provider, an AS, or an eavesdropper on a wireless network. The goal of the adversary is to i) read the content of messages, and to ii) impersonate *peer* devices.

We assume that

- All peers are honest and do not collaborate with the network adversary.
- A Peer (Alice) can send a single QR-code sized *invite code* in an *out-of-band channel* to another peer (Bob). The attacker can not observe or modify the invite code.

The SecureJoin protocols allow *peer* devices to establish *guaranteed end-to-end encryption* that is resistant to machine-in-the-middle attacks by preventing the adversary from reading messages or impersonating honest peers.

[5] https://delta.chat
[6] https://autocrypt.org
[7] https://autocrypt.org
[8] https://countermitm.readthedocs.io/en/latest/

Passive adversaries such as message transport providers can still learn which peers communicate with each other, at what time and the approximate size of the messages.

An adversary who can observe Alice's invite code in the out-of-band channel can perform impersonation attacks. Additional measures can relax the security requirements for the out-of-band channel to also work under a threat of observation.

1.2 Disadvantages of other key-verification techniques

An important aspect of guaranteed end-to-end (e2e) encryption is the verification of a peer's key. In many existing e2e-encrypting messengers like Signal or Element, users perform key verification by triggering two fingerprint verification workflows: each of the two peers shows and reads the other's key fingerprint through a trusted channel (often a QR code show+scan).

We observe the following issues with these schemes:

- The schemes require that both peers start the verification workflow to assert that both of their encryption keys are not manipulated. Such double work has an impact on usability.
- In the case of a group, every peer needs to verify keys with each group member to be able to assert that messages are coming from and are encrypted to the true keys of members. A peer that joins a group of size N must perform N verifications. Forming a group of size N therefore requires $N(N-1)/2$ verifications in total. Thus this approach is impractical even for moderately sized groups.
- Users often fail to distinguish Lost/Reinstalled Device events from Machine-in-the-Middle (MITM) attacks, see for example When Signal hits the Fan[9].

1.3 Integrating key verification with general workflows

In *Securing communications against network adversaries* (page 5) we describe new protocols that aim to resolve these issues, by integrating key verification into existing messaging use cases:

- the *Setup Contact protocol* (page 6) allows a user, say Alice, to establish a verified contact with another user, say Bob. At the end of this protocol, Alice and Bob know each other's contact information and have verified each other's keys. To do so, Alice sends an *invite code* using the second channel to Bob (for example, by showing it as a QR code). The invite code transfers not only the key fingerprint, but also contact information (e.g., email address). After receiving the second-channel invite code, Alice's and Bob's clients communicate via the regular messaging channel to 1) exchange Bob's key and contact information and 2) to verify each other's keys. Note that this protocol only uses one out-of-band message requiring involvement of the user. All other messages are sent in the regular channel potentially controlled by a network adversary. Note that this protocol only requires one *invite code* to be

[9] https://eurousec.secuso.org/2016/presentations/WhenSignalHitsFan.pdf

transferred un-observed. All other messages are exchanged on the regular messaging channel controled by the network adversary.

- the *Verified Group protocol* (page 11) enables a user to invite another user to join a verified group as a new member. This protocol builds on top of the previous protocol. The “joining” peer first establishes verified contact with the inviter, and the inviter then announces the joiner as a new member. At the end of this protocol, the “joining” peer has learned the keys of all members of the group.

  Any group member may invite new members. By introducing members in this incremental way, a group of size N requires only $N - 1$ verifications overall to ensure that a network adversary can not compromise end-to-end encryption between group members. If one group member loses her key (e.g. through device loss), she must re-join the group via invitation of the remaining members of the verified group.

1.4 Known Limitations and Issues

- The verification of the fingerprint only checks the current keys. Since protocols do not store any historical information about keys, the verification can not detect if there was a past temporary MITM-exchange of keys (say the network adversary exchanged keys for a few weeks but changed back to the “correct” keys afterwards).

2 Securing communications against network adversaries

To withstand network adversaries, peers must verify each other's keys to establish guaranteed e2e-encrypted communication. In this section we describe protocols to securely setup a contact and to securely add a user to a group.

Establishing guaranteed e2e-encrypted communication is particularly difficult in group communications where more than two peers communicate with each other. Existing messaging systems usually require peers to verify keys with every other peer to assert that they have guaranteed e2e-encryption. This is highly unpractical. First, the number of verifications that a single peer must perform becomes too costly even for small groups. Second, a device loss will invalidate all prior verifications of a user. Rejoining the group with a new device (and a new key) requires redoing all the verification, a tedious and costly task. Finally, because key verification is not automatic – it requires users' involvement – in practice very few users consistently perform key verification.

Key consistency schemes do not remove the need of key verification. It is possible to have a group of peers which each see consistent email-addr/key bindings from each other, yet a peer is consistently isolated by a network adversary performing a machine-in-the-middle attack. It follows that each peer needs to verify with at least one other peer to assure that there is no isolation attack.

A known approach to reduce the number of necessary key verifications is the web of trust. This approach requires a substantial learning effort for users to understand the underlying concepts, and is hardly used outside specialist circles. Moreover, when using OpenPGP, the web of trust is usually interacting with OpenPGP key servers. These servers make the signed keys widely available, effectively making the social "trust" graph public. Both key servers and the web of trust have reached very limited adoption.

Autocrypt was designed to not rely on public key servers, nor on the web of trust. It thus provides a good basis to consider new key verification approaches. To avoid the difficulties around talking about keys with users, we implemented new protocols which perform key verification as part of other workflows, namely:

- setting up a contact between two individuals who meet physically, and
- setting up a group with people who you meet or have met physically.

These new workflows require *administrative* messages to support the authentication and security of the key exchange process. These administrative messages are sent between devices, but are not shown to the user as regular messages.

The additional advantage of using administrative messages is that they significantly improve usability by reducing the overall number of user actions.

Note that automated processing of administrative messages opens up a new attack vector: malfeasant peers can try to inject administrative messages in order to impersonate another user or to learn if a particular user is online. ..

> TODO: Link to PR/issue about prevent-online-leak: https://github.com/deltachat/deltachat-core-rust/pull/4932 TODO: Add new attack vectors to known limitations in

summary.rst

All protocols that we introduce in this section are *decentralized*. They describe how peers (or their devices) can interact with each other, without having to rely on services from third parties. Our verification approach thus fits into the Autocrypt key distribution model which does not require extra services from third parties either.

While Autocrypt Level 1 focuses on passive attacks such as sniffing the mail content by a provider, active attacks are outside of the scope and can be carried out automatically by replacing Autocrypt headers.

With the SecureJoin protocols we aim to increase the costs of active attacks by introducing a second out-of-band channel and using it to verify the Autocrypt headers transmitted in-band.

We consider targeted active attacks against these protections feasible. However they will require coordinated attacks based for example on infiltrators or real time CCTV footage.

We believe that the ideas explained here make automated mass surveillance prohibitively expensive with a fairly low impact on usability.

2.1 Setup Contact protocol

The goal of the Setup Contact protocol is to allow two peers to conveniently establish secure contact: exchange both their e-mail addresses and cryptographic identities in a verified manner. This protocol is re-used as a building block for the *verified-group* (page 11) protocol.

After running the Setup Contact protocol, both peers will learn the cryptographic identities (i.e., the keys) of each other or else both get an error message. The protocol is safe against active attackers that can modify, create and delete messages.

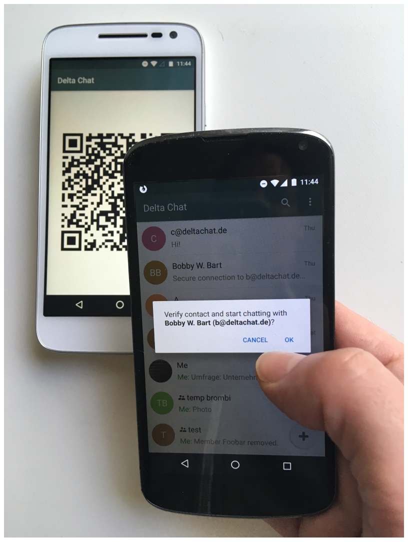


Fig. 1: Setup Contact protocol step 2.

The protocol follows a single simple UI workflow: A peer "shows" an invite code once that is then "read" by the other peer through an out-of-band channel. This means that, as opposed to current fingerprint verification workflows, the protocol only runs once instead of twice, yet results in the two peers having verified keys of each other.

Between mobile phones, showing and scanning a QR code constitutes an out-of-band channel, but transferring data via USB, Bluetooth, WLAN or other network-adversary resilient messengers is possible as well.

Recall that we assume that our active attacker *cannot* observe or modify data transferred via the out-of-band channel.

An attacker who can alter messages but has no way of reading or manipulating the out-of-band channel can prevent the verification protocol from completing successfully by dropping or altering messages.

An attacker who can compromise both channels can inject wrong key material and convince the peer to verify it.

Fig. 2: UI and administrative message flow of contact setup

Here is a conceptual step-by-step example of the proposed UI and administrative message workflow for establishing a secure contact between two contacts, Alice and Bob.

1. Alice sends a invite code to Bob via the out-of-band channel.

   a) The invite code consists of:

   - Alice's Openpgp4 public key fingerprint `Alice_FP`, which acts as a commitment to the Alice's Autocrypt key, which she will send later in the protocol,
   - Alice's e-mail address (both name and routable address),
   - a challenge `INVITENUMBER` of at least 8 bytes. This challenge is used by Bob's device in step 2b to prove to Alice's device that it is the device that the invite code was shared with. Alice's device uses this information in step 3 to automatically accept Bob's contact request. This is in contrast with most messaging apps where new contacts typically need to be manually confirmed.
   - a second challenge `AUTH` of at least 8 bytes which Bob's device uses in step 4 to authenticate itself against Alice's device.

   b) In the `tokens` SQL table, Alice's device will keep track of: - the namespace (`INVITENUMBER` or `AUTH`) - if this is a *verified-group* (page 11) invite: the group chat id - the token itself - the time the contact verification was initiated.

2. Bob receives the invite code and

   a) If Bob's device already knows a key with the fingerprint `Alice_FP` that belongs to Alice's e-mail address the protocol continues with 4b)

    b) otherwise Bob's device sends a cleartext "vc-request" message to Alice's e-mail address, adding the `INVITENUMBER` from step 1 to the message. Bob's device automatically includes Bob's Autocrypt key in the message.

3. Alice's device receives the "vc-request" message. As with any incoming Autocrypt message, she saves Bob's public key.

   a) She looks up the invite code for the `INVITENUMBER`. If the `INVITENUMBER` does not match then Alice terminates the protocol.

    b) (removed)

    c) (removed)

   d) She uses this key to create an encrypted "vc-auth-required" message containing her own Autocrypt key, which she sends to Bob.

4. Bob receives the "vc-auth-required" message, decrypts it, and verifies that Alice's Autocrypt key matches `Alice_FP`.

    a) If verification fails, Bob gets a screen message "Cannot establish guaranteed end-to-end encryption with Alice" and the protocol terminates.

    b) Otherwise Bob's device sends back a 'vc-request-with-auth' encrypted message whose encrypted part contains Bob's own key fingerprint `Bob_FP` and the second challenge `AUTH` from step 1.

5. Alice decrypts Bob's 'vc-request-with-auth' message

   a) and verifies that Bob's Autocrypt key matches `Bob_FP` that the invite has not expired and that the transferred `AUTH` matches the one from step 1.

   b) If any verification fails, Alice's device signals "Cannot establish guaranteed end-to-end encryption with Bob" and the protocol terminates.

6. If the verification succeeds on Alice's device

    a) shows "Secure contact with Bob <bob-adr> established".

    b) sends Bob a "vc-contact-confirm" message.

7. Bob's device receives "vc-contact-confirm" and shows "Secure contact with Alice <alice-adr> established".

At the end of this protocol, Alice has learned and validated the contact information and Autocrypt key of Bob, the person to whom she sent the invite code. Moreover, Bob has learned and validated the contact information and Autocrypt key of Alice, the person who sent the invite code to Bob.

2.1.1 Requirements for the underlying encryption scheme

The Setup Contact protocol requires that the underlying encryption scheme is non-malleable. Malleability means the encrypted content can be changed in a deterministic way. Therefore with a malleable scheme an attacker could impersonate Bob: They would add a different autocrypt key in Bob's vc-request message (step 2.b) and send the message along without other changes. In step 4.b they could then modify the encrypted content to include their own keys fingerprint rather than `Bob_FP`.

In the case of OpenPGP non-malleability is achieved with Modification Detection Codes (MDC - see section 5.13 and 5.14 of RFC 4880). Implementers need to make sure to verify these and treat invalid or missing MDCs as an error. Using an authenticated encryption scheme prevents these issues and is therefore recommended if possible.

2.1.2 An active attacker cannot break the security of the Setup Contact protocol

Recall that an active attacker can read, modify, and create messages that are sent via a regular channel. The attacker cannot observe or modify the invite code that Alice sends via the out-of-band channel. We argue that such an attacker cannot break the security of the Setup Contact protocol, that is, the attacker cannot impersonate Alice to Bob, or Bob to Alice.

Assume, for a worst-case scenario, that the adversary knows the public Autocrypt keys of Alice and Bob. At all steps except step 1, the adversary can drop messages. Whenever the adversary drops a message, the protocol fails to complete. Therefore, we do not consider dropping of messages further.

1. The adversary cannot impersonate Alice to Bob, that is, it cannot replace Alice's key with a `Alice-MITM` key known to the adversary. Alice sends her key to Bob in the encrypted "vc-auth-required" message (step 3). The attacker can replace this message with a new "vc-auth-required" message, again encrypted against Bob's real key, containing a fake `Alice-MITM` key. However, Bob will detect this modification during step 4a, because the fake `Alice-MITM` key does not match the fingerprint `Alice_FP` that Alice sent to Bob in the invite code. (Recall that the invite code is transmitted via the out-of-band channel the adversary cannot modify.)

2. The adversary also cannot impersonate Bob to Alice, that is, it cannot replace Bob's key with a `Bob-MITM` key known to the adversary. The cleartext "vc-request" message, sent from Bob to Alice in step 2, contains Bob's key. To impersonate Bob, the adversary must substitute this key with the fake `Bob-MITM` key.

   In step 3, Alice cannot distinguish the fake key `Bob-MITM` inserted by the adversary from Bob's real key, since she has not seen Bob's key in the past. Thus, she will follow the protocol and send the reply "vc-auth-required" encrypted with the key provided by the adversary.

   We saw in the previous part that if the adversary modifies Alice's key in the "vc-auth-required" message, then this is detected by Bob. Therefore, it forwards the "vc-auth-required" message unmodified to Bob.

Since `Alice_FP` matches the key in “vc-auth-required”, Bob will in step 4b send the “vc-request-with-auth” message encrypted to Alice’s true key. This message contains Bob’s fingerprint `Bob_FP` and the challenge `AUTH`.

Since the message is encrypted to Alice’s true key, the adversary cannot decrypt the message to read its content. There are now three possibilities for the attacker:

- The adversary modifies the “vc-request-with-auth” message to replace `Bob_FP` (which it knows) with the fingerprint of the fake `Bob-MITM` key. However, the encryption scheme is non-malleable, therefore, the adversary cannot modify the message, without being detected by Alice.
- The adversary drops Bob’s message and creates a new fake message containing the fingerprint of the fake `Bob-MITM` key and a guess for the challenge `AUTH`. The adversary cannot learn the challenge `AUTH`: it cannot observe the invite code transmitted via the out-of-band in step 1, and it cannot decrypt the message “vc-request-with-auth”. Therefore, this guess will only be correct with probability 2^{-64}. Thus, with overwhelming probability Alice will detect the forgery in step 5, and the protocol terminates without success.
- The adversary forwards Bob’s original message to Alice. Since this message contains Bob’s key fingerprint `Bob_FP`, Alice will detect in step 5 that Bob’s “vc-request” from step 3 had the wrong key (Bob-MITM) and the protocol terminates with failure.

2.1.3 Replay attacks and conflicts

Alice’s device records the time a contact verification was initiated. It also verifies it has not expired and clears the data after completion. This prevents replay attacks. Replay attacks could be used to make Alice’s device switch back to an old compromised key of Bob.

Limiting an invite to a single use reduces the impact of a QR-code being exposed to an attacker: If the attacker manages to authenticate faster than Bob they can impersonate Bob to Alice. However Bob will see an error message. If the QR-code could be reused the attacker could successfully authenticate. Alice would have two verified contacts and Bob would not see any difference to a successful connection attempt.

Furthermore a compromise of Bob’s device would allow registering other email addresses as verified contacts with Alice.

2.1.4 Business Cards

QR-codes similar to the ones used for verified contact could be used to print on business cards.

Since business cards are usually not treated as confidential they can only serve to authenticate the issuer of the business card (Alice) and not the recipient (Bob).

However as discussed on the messaging@moderncrypto mailing list[10] the verification of a short code at the end of the protocol can extend it to also protect against leakage of the QR-code. This may also be desirable for users who face active surveillance in real life and therefor cannot assume that scanning the QR-code is confidential.

2.1.5 Open Questions

- (how) can messengers such as Delta.chat make "verified" and "opportunistic" contact requests be indistinguishable from the network layer?
- (how) could other mail apps such as K-9 Mail / OpenKeychain learn to speak the "setup contact" protocol?

2.2 Verified Group protocol

We introduce a new secure **verified group** that enables secure communication among the members of the group. Verified groups provide these simple to understand properties:

1. All messages in a verified group are end-to-end encrypted and secure against active attackers. In particular, neither a passive eavesdropper, nor an attactive network attacker (e.g., capable of man-in-the-middle attacks) can read or modify messages.
2. There are never any warnings about changed keys (like in Signal) that could be clicked away or cause worry. Rather, if a group member loses her device or her key, then she also looses the ability to read from or write to the verified group. To regain access, this user must join the group again by finding one group member and perform a "secure-join" as described below.

2.2.1 Verifying a contact to prepare joining a group

The goal of the secure-join protocol is to let Alice make Bob a member (i.e., let Bob join) a verified group of which Alice is a member. Alice may have created the group or become a member prior to the addition of Bob.

In order to add Bob to the group Alice has to verify him as a contact if she has not done so yet. We use this message exchange to also ask Bob whether he agrees to becoming part of the group.

[10] https://moderncrypto.org/mail-archive/messaging/2018/002544.html

The protocol re-uses the first five steps of the *setup-contact* (page 6) protocol so that Alice and Bob verify each other's keys. To ask for Bob's explicit consent we indicate that the messages are part of the verified group protocol, and include the group's identifier in the metadata part of the verified-group invite code.

More precisely:

- in step 1 Alice adds the metadata `INVITE=<groupname>`. Where `<groupname>` is the name of the group `GROUP`.
- in step 2 Bob manually confirms he wants to join `GROUP` before his device sends the `vc-request` message. If Bob declines processing aborts.
- in step 5 Alice looks up the metadata associated with the `INVITENUMBER`. If Alice sees the `INVITE=<groupname>` but is not part of the group anymore she aborts the joining process (without sending another message).

If no failure occurred up to this point, Alice and Bob have verified each other's keys, and Alice knows that Bob wants to join the group `GROUP`.

The protocol then continues as described in the following section (steps 6 and 7 of the *setup-contact* (page 6) are not used).

2.2.2 Joining a verified group ("secure-join")

In order to add Bob to a group Alice first needs to make sure she has a verified key for Bob. This is the case if Bob already was a verified contact or Alice performed the steps described in the previous section.

Now she needs to inform the group that Bob should be added. Bob needs to confirm everything worked:

a. Alice broadcasts an encrypted "vg-member-setup" message to all members of `GROUP` (including Bob), gossiping the Autocrypt keys of all members (including Bob).

b. Bob receives the encrypted "vg-member-setup" message. Bob's device verifies:

   - The encryption and Alice's signature are intact.
   - Alice may invite Bob to a verified group. That is she is a verified contact of Bob.

   If any of the checks fail processing aborts. Otherwise the device learns all the keys and e-mail addresses of group members. Bob's device sends a final "vg-member-setup-received" message to Alice's device. Bob's device shows "You successfully joined the verified group `GROUP`".

c. Any other group member that receives the encrypted "vg-member-setup" message will process the gossiped key through autocrypt gossip mechanisms. In addition they verify:

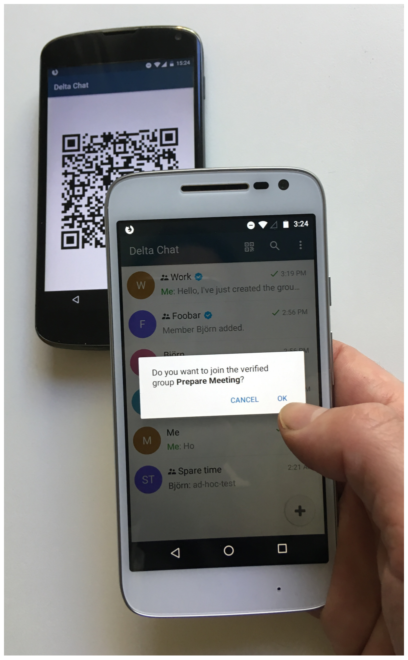
   - The encryption and Alice's signature are intact.

- They are themselves a member of `GROUP`.
- Alice is a member of `GROUP`.

If any of the checks fail processing aborts. Otherwise they will add Bob to their list of group members and mark the gossiped key as verified in the context of this group.

d. Alice's device receives the "vg-member-setup-received" reply from Bob and shows a screen "Bob <email-address> securely joined group `GROUP`"

Bob and Alice may now both invite and add more members which in turn can add more members. The described secure-join workflow guarantees that all members of the group have been verified with at least one member. The broadcasting of keys further ensures that all members are fully connected.



Me: Hello, I've just created the grou...

Do you want to join the verified group **Prepare Meeting**?

Fig. 3: Join-Group protocol at step 2 with https://delta.chat.

2.2.3 Strategies for verification reuse

Since we retrieve keys for verified groups from peers we have to choose whether we want to trust our peers to verify the keys correctly.

One of the shortcomings of the web of trust is that it's mental model is hard to understand and make practical use of. We therefore do not ask the user questions about how much they trust their peers.

Therefore two strategies remain that have different security implications.

Delta Chat chose the "Ignoring infiltrators, focusing on message transport attacks first" strategy.

- **Restricting verification reuse across groups** Since we share the content of the group with all group members we can also trust them to verify the keys used for the group.

  If they wanted to leak the content they could do so anyway.

  However if we want to reuse keys from one verified group to form a different one the peer who originally verified the key may not be part of the new group.

  If the verifier is "malicious" and colludes with an attacker in a MITM position, they can inject a MITM key as the verified key. Reusing the key in the context of another group would allow MITM attacks on that group.

  This can be prevented by restricting the invitation to verified groups to verified contacts and limiting the scope of keys from member-added messages to the corresponding group.

- **Ignoring infiltrators, focusing on message transport attacks first** One may also choose to not consider advanced attacks in which an "infiltrator" peer collaborates with an evil provider to intercept/read messages.

  In this case keys can be reused across verified groups. Active attacks from an adversary who can only modify messages in the regular transport channel are still impossible.

  A malicious verified contact may inject MITM keys. Say Bob when adding Carol as a new member, sends a prepared MITM key. We refer to this as a Bob in the middle attack to illustrate that a peer is involved in the attack.

  We note, that Bob, will have to sign the message containing the gossip fake keys.

  Trusting all peers to verify keys also allows faster recovery from device loss. Say Alice lost her device and Bob verified the new key. Once Bob announced the new key in a verified group including Carol Carol could send the key to further verified groups that Bob is not part of.

2.2.4 Dealing with key loss and compromise

If a user looses their device they can setup a new device and regain access to their inbox. However they may loose their secret key.

They can generate a new key pair. Autocrypt will distribute their new public key in the Autocrypt headers and opportunistic encryption will switch to it automatically.

Verified groups will remain unreadable until the user verifies a contact from that group. Then the contact can update the key used in the group. This happens by sending a "vg-member-setup" message to the group. Since the email address of that user remains the same the old key will be replaced by the new one.

Implementers may decide whether the recipients of such key updates propagate them to other groups they share with the user in question. If they do this will speed up the recovery from device loss. However it also allows Bob-in-the-middle attacks that replace the originally verified keys. So the decision needs to be based on the threat model of the app and the strategy picked for verification reuse

If a key is known or suspected to be compromised more care needs to be taken. Since network attackers can drop messages they can also drop the "vg-member-setup" message that was meant to replace a compromised key. A compromised key combined with a network attack breaks the security of both channels. Recovering from this situation needs careful consideration and goes beyond the scope of our current work.

2.2.5 Notes on the verified group protocol

- **More Asynchronous UI flow**: All steps after 2 (the sending of administrative messages) could happen asynchronously and in the background. This might be useful because e-mail providers often delay initial messages ("greylisting") as mitigation against spam. The eventual outcomes ("Could not establish verified connection" or "successful join") can be delivered in asynchronous notifications towards Alice and Bob. These can include a notification "verified join failed to complete" if messages do not arrive within a fixed time frame. In practise this means that secure joins can be concurrent. A member can show the "Secure Group invite" to a number of people. Each of these peers scans the message and launches the secure-join. As 'vc-request-with-auth' messages arrive to Alice, she will send the broadcast message that introduces every new peer to the rest of the group. After some time everybody will become a member of the group.
- **Leaving attackers in the dark about verified groups**. It might be feasible to design the step 3 "secure-join-requested" message from Bob (the joiner) to Alice (the inviter) to be indistinguishable from other initial "contact request" messages that Bob sends to Alice to establish contact. This means that the provider would, when trying to substitute an Autocrypt key on a first message between two peers, run the risk of **immediate and conclusive detection of malfeasance**. The introduction of the verified group protocol would thus contribute to securing the e-mail encryption eco-system, rather than just securing the group at hand.
- **Sending all messages through alternative channels**: instead of being relayed through the provider, all messages from step 2 onwards could be transferred via Bluetooth or WLAN. This way, the full invite/join protocol would be completed on a different channel. Besides increasing the security of the joining, an additional advantage is that the provider would not gain knowledge about verifications.
- **Non-messenger e-mail apps**: instead of groups, traditional e-mail apps could possibly offer the techniques described here for "secure threads".

2.2.6 Autocrypt and verified key state

Verified key material – whether from verified contacts or verified groups – provides stronger security guarantees then keys discovered in Autocrypt headers.

At the same time opportunistic usage of keys from autocrypt headers provides faster recovery from device loss.

Therefore the address-to-key mappings obtained using the verification protocols should be stored separately and in addition to the data stored for the normal Autocrypt behaviour.

Verified contacts and groups offer a separate communication channel from the opportunistic one.

We separated the two concepts but they can both be presented to the user as 'Verified Groups'. In this case the verified contact is a verified group with two members.

This allows the UI to feature a verified group and the 'normal' opportunistic encryption with the same contact.

The verified group prevents key injection through Autocrypt headers. In the case of device loss the user can fall back to the non-verified contact to ensure availability of a communication channel even before the next verification has taken place.