Cryptographic Analysis of Delta Chat*

Yuanming Song
ETH Zurich

Lenka Mareková
ETH Zurich

Kenneth G. Paterson
ETH Zurich

Abstract

We analyse the cryptographic protocols underlying Delta Chat, a decentralised messaging application which uses e-mail infrastructure for message delivery. It provides end-to-end encryption by implementing the Autocrypt standard and the SecureJoin protocols, both making use of the OpenPGP standard. Delta Chat's adoption by categories of high-risk users such as journalists and activists, but also more generally users in regions affected by Internet censorship, makes it a target for powerful adversaries. Yet, the security of its protocols has not been studied to date. We describe five new attacks on Delta Chat in its own threat model, exploiting cross-protocol interactions between its implementation of SecureJoin and Autocrypt, as well as bugs in rPGP, its OpenPGP library. The findings have been disclosed to the Delta Chat team, who implemented fixes.

1 Introduction

Delta Chat[1] is an open-source decentralised messaging application based on e-mail infrastructure. It can be seen as a custom-built e-mail client: users log into Delta Chat with their existing e-mail accounts, and messages in Delta Chat are specially formatted e-mails. It does not have a central server, since users can choose different e-mail providers or even host their own e-mail servers. It provides end-to-end encryption by implementing the Autocrypt standard [58] and the SecureJoin protocols [23], both based on a subset of the OpenPGP standard [35]. It functions in two modes, with different security: *verified chats* guarantee end-to-end encryption, while *unverified chats* are not intended to be secure against active network attackers, i.e. they are vulnerable to machine-in-the-middle (MITM) attacks, and may not even be encrypted.

While Delta Chat does not explicitly advertise itself as suitable for high-risk users, these users likely make up a noticeable part of Delta Chat's user base. For example, Delta Chat's website lists feedback from people in regions with strict Internet governance [22]. Delta Chat works with activist groupings and researchers to improve the usability and security of the application for high-risk users, and has produced several need-finding reports by engaging with journalists and activists in Ukraine [33, 34] and elsewhere [32].

Delta Chat has been supported or recommended by several organisations in the human rights context. Among other funding sources, Delta Chat has received support from the U.S. Bureau of Democracy, Human Rights and Labor to make Delta Chat "more resilient and secure in places often affected by internet censorship and shutdowns" [20]. Front Line Defenders, a human rights organisation, recommended Delta Chat for secure communication during COVID-19 [27]; eQualitie, a digital security organisation, recommends Delta Chat to Ukrainian users in the ongoing Russia-Ukraine war [53].

Despite being perceived as secure and engaging with high-risk users, the security of Delta Chat is still largely unclear, especially for its cryptographic components and protocol implementations. Further, while its position as both an instant messenger and an e-mail client ensures easier adoption, it also brings more complexity to its protocols, thereby enlarging its attack surface. The Autocrypt and SecureJoin protocols have received considerable input from academia [49], but have never been formally analysed. This is especially concerning since these protocols use OpenPGP in non-standard ways, and the Delta Chat team has made substantial changes to the SecureJoin protocols. Further, the OpenPGP library used by Delta Chat, rPGP, has received relatively little attention compared to other OpenPGP libraries.

Recent works have uncovered serious security flaws in several messaging applications deploying custom cryptographic protocols such as Bridgefy [3,6], Matrix [4] and Threema [51]. In the last two, flaws were found after the application in question was "vetted" via security audits, showing that audits alone cannot establish a protocol as secure and trustworthy. Indeed, Delta Chat too has undergone two audits that covered its use of cryptography [56, 57]; as we will see, this was not sufficient to prevent attacks.

*This is the full version of a work to appear at USENIX 2024.

[1] https://delta.chat/

1.1 Contributions

We have analysed the core cryptographic protocols used by Delta Chat, discovering five new attacks:

1. **Gossip key injection.** An attacker can send an e-mail to the target such that the target will replace the public key of a given contact of the target with an attacker-controlled key and mark it as verified.

2. **Group member removal.** An attacker can remove arbitrary members from a verified group as if they were removed by other members by adding a carefully constructed plaintext e-mail header to a normal group message.

3. **Synchronisation forgery.** An attacker capable of spoofing self-addressed e-mails for the target can trivially forge synchronisation messages. This allows the attacker to verify themselves to the target or change chat statuses.

4. **InsecureJoin observer.** An attacker can exploit the fact that clients observe self-sent SecureJoin protocol messages from other devices to trick the target into marking an attacker-controlled key as verified.

5. **Autocrypt Setup forgery.** An attacker can forge an Autocrypt Setup Message, which is used to transfer a user's secret key between different e-mail clients. This enables the attacker to make the target use a key known by the attacker.

Overall, our analysis demonstrates that Delta Chat users were vulnerable to: a MITM attacker reading and modifying their messages even in verified chats, being impersonated to other Delta Chat users (or vice versa), their chat statuses and setup being modified, being deceived to use an attacker-controlled OpenPGP key and consequently losing most security guarantees, and denial-of-service attacks.

1.2 Related Work

No prior works to date provide an in-depth security analysis of Delta Chat or its specialised protocols. ClaimChain [44], a part of the CounterMITM [50] specification which served as the basis for SecureJoin, is a solution for privacy-preserving key distribution for Autocrypt, but it was never implemented in Delta Chat. A recent work [29] proposed to extend the OpenPGP library used by Delta Chat with post-quantum primitives, but did not include a security analysis.

OpenPGP itself has a long history in the literature as evidenced by EFAIL [52], attacks on signatures [48] and ElGamal implementations [26], format oracle attacks [41] and key overwriting attacks [17]. It has also been criticised for its lack of usability [55, 61] and wider standardisation and deployment issues [37]. Yet, efforts are underway for updating the cryptographic building blocks of OpenPGP [62].

Delta Chat has undergone four independent security audits [20], and is planning a fifth audit in 2024 [39]. Two of the audits covered the cryptographic components of Delta Chat, including one in 2019 on its OpenPGP and RSA libraries [56], and another in 2020 on its core Rust libraries [57], both conducted by Include Security.

Our work fits within the larger area of cryptographic study of secure messaging protocols. A number of works show attacks on messaging apps [3, 4, 6, 7, 51], while other works provide security proofs, largely focusing on Signal-based protocols [9, 15, 24], though attacks may still be possible [25]. From a wider perspective, analysis of TLS 1.2 has provided both security proofs [43, 47] and practical attacks [1, 8, 13].

In Signal, verification of user keys is more invisible than in Delta Chat, relegated to an optional feature of verifying "safety numbers". Recent works have focused on both its usability aspects [59] and enhancing its security guarantees [30, 31]. For group communication, Messaging Layer Security (MLS) [14] could provide an alternative, provably secure design [10, 11, 18]; however, implementing it in a decentralised fashion remains an open problem [12].

Notably, Delta Chat's need-finding reports [32–34] uncovered many of the information security practices of higher-risk users that were later replicated in a different context [2].

1.3 Methodology

We conducted a security analysis on the cryptography used in Delta Chat, covering Delta Chat's cryptographic components as well as Delta Chat's implementation of the Autocrypt and SecureJoin protocols. We manually analysed Delta Chat's core Rust implementation[2] as well as the relevant parts of rPGP,[3] Delta Chat's OpenPGP library: we analysed version 1.132.0 of the Delta Chat core library,[4] released on 6 December 2023, and version 0.10.2 of the rPGP library,[5] released on 24 July 2023. We studied the specifications of OpenPGP [35], Autocrypt [58], and SecureJoin [23] and compared them with their implementations in Delta Chat and rPGP.

We implemented our attacks using the Python bindings for the Delta Chat core library.[6] We instantiated eavesdroppers and network attackers as malicious SMTP proxies, and we used a local testing e-mail server for attack simulation. We verified our proof-of-concept attacks on Delta Chat core versions 1.132.0 and 1.133.2[7] and Delta Chat Desktop 1.42.2.

However, we did not specifically analyse how Delta Chat applications in different platforms use the core library, and we did not analyse the security of experimental features such as location streaming. Since Delta Chat and its dependencies

[2] https://github.com/deltachat/deltachat-core-rust

[3] https://github.com/rpgp/rpgp

[4] https://github.com/deltachat/deltachat-core-rust/releases/tag/v1.132.0

[5] https://crates.io/crates/pgp/0.10.2

[6] https://pypi.org/project/deltachat/

[7] The latest core version available to us at the time.

are open-source, we did not perform any reverse engineering for our analysis. Our analysis and tests did not involve any real Delta Chat users or target the Delta Chat infrastructure such as Delta Chat bots[8] and chatmail services.[9]

1.4 Responsible Disclosure

We disclosed our findings to the Delta Chat development team on 2 February 2024, suggesting a 90-day disclosure period. They acknowledged receipt and agreed to a coordinated disclosure on 7 February 2024. The team implemented fixes for all of the main attacks as well as for a number of additional issues identified during our analysis. Some of the fixes were implemented in rPGP. The fixes were released to end users as part of Delta Chat application versions 1.44 on 12-16 March 2024, with a public disclosure describing the issues on 25 March 2024 in order to allow time for users to update [21].

1.5 Overview of the Paper

We describe Delta Chat's threat model in Section 2 and its architecture in Section 3. We describe our attacks in Section 4 and discuss the implications of our findings in Section 5.

2 Threat Model

Delta Chat provides different levels of security for different types of chats. *Verified chats* provide guaranteed end-to-end encryption with the SecureJoin protocols, protecting against network attackers that can actively modify messages, such as malicious e-mail servers. *Unverified chats* are opportunistically end-to-end encrypted via Autocrypt, protected only against passive eavesdropping attackers. Hence, for unverified chats, MITM attacks are outside of the threat model [39]; it is also possible to use them to send unencrypted messages. We define our threat model such that the attacker targeting different types of chats has different capabilities and goals.

Delta Chat does not offer advanced properties such as forward secrecy and post-compromise security, and verified chats do not aim to protect against insider attacks. Our threat model follows Delta Chat's security claims and assumptions, thus only capturing a subset of the attackers typically considered for other end-to-end encrypted messaging applications.

Capabilities. We consider attackers with one or more of the following capabilities, ordered roughly by their strength. Note that different capabilities may overlap with each other.

Sending e-mails. The attacker knows the target's e-mail address and is able to send e-mails to the target without being flagged by the spam detector or blocked by the target. In Delta Chat, it is possible to receive e-mails from non-contacts, in which case the e-mails appear as "message requests" [20].

Knowledge of public keys. The attacker knows the target's public key. This capability is trivial for eavesdropping attackers since they can observe the Autocrypt headers attached to all of the target's outgoing e-mails. The attacker may also learn the target's public key by tricking the target into sending an e-mail to an attacker-controlled e-mail address, or through the Autocrypt key gossip mechanism (Section 3.1.3).

Eavesdropping. The attacker is able to observe messages on the network, including the target's incoming and outgoing e-mails; this is assumed in the Autocrypt standard [58].

E-mail spoofing. The attacker is able to create and send e-mails to the target with the `From` header arbitrarily chosen by the attacker, e.g. through an insecure SMTP server [45]. A special case is spoofing an e-mail as if it were sent from the target to themselves, where both the `From` and `To` headers represent the target's e-mail address.

Control of network messages. The attacker can read, create, modify, and delete messages. The SecureJoin protocols [23, Section 1.1] as well as Delta Chat [20] consider such a network attacker in their threat model.

Limiting assumptions. Attackers targeting unverified chats are assumed to not spoof e-mails or modify network messages.

We assume that users can share their QR invite codes via a secure out-of-band channel that cannot be observed or modified by the attacker [23, Section 1.1].

All verified contacts of the target are honest, and no infiltrators colluding with the attacker are present in the verified groups where the target is a member [23, Section 2.2.3]. Note that this assumption does not exclude the attacker from communicating with the target through unverified chats, or establishing themselves as a verified contact or a group member by exploiting vulnerabilities in Delta Chat.

Goals. The Autocrypt standard does not specify the attacker's goals, but it can be inferred that Autocrypt intends to protect the content of encrypted messages. The SecureJoin protocols specify the attacker's goals as "i) read the content of messages, and to ii) impersonate peer devices" [23, Section 1.1]. Finally, Delta Chat states that messages in verified chats "can not be read or altered by compromised e-mail servers or Internet providers" [20]. Thus, the attacker's goal is to break the confidentiality or integrity of encrypted messages, break authenticity by forging messages in verified chats, and break verification guarantees in Delta Chat. The latter includes the attacker getting themselves or some honest peer to be verified by the target under a key controlled by the attacker, and changing the setup of verified chats.[10]

[8] https://bots.delta.chat/

[9] https://delta.chat/en/2023-12-13-chatmail

[10] Having the target verify a key controlled by the attacker allows the attacker to perform a "Bob in the middle" attack [23, Section 2.2.3] to the target as a malicious insider and break all previous security properties. Changing the verified chat setup qualifies as breaking integrity or authenticity.

We do not consider removing, duplicating, or reordering messages as among the goals of the attacker, even though they are trivial for network attackers and can become real threats.

3 Delta Chat

In this section, we introduce Autocrypt (Section 3.1) as well as the SecureJoin protocols (Section 3.2), which together form the cryptographic core of Delta Chat.

Delta Chat relies on a subset of the OpenPGP standard for end-to-end encryption that Delta Chat claims to be secure [20], implemented in the rPGP library. We summarise the parts of OpenPGP relevant to our analysis in Appendix A.

User's perspective. Upon installation, Delta Chat prompts the user to log into their e-mail account or import an existing Delta Chat account from another device. The user can add a contact by scanning a QR code of another user, inputting an e-mail address or being part of the same group chat. When the user sends an encrypted message, it is shown with a small padlock. If the encryption material for a contact is considered *verified*, chats with that contact are shown with a green checkmark. Verification needs little effort from the user: a contact is verified when added using their invite code or from a verified chat, and a chat is verified if all members are verified.

Public keys. Delta Chat associates each contact or *peer* with several OpenPGP keys. In the context of Autocrypt, these will be referred to as the peer's *Autocrypt (public) key* and the peer's *gossip key*. In the context of SecureJoin, we will also define *verified keys*. The functions of these keys will be explained in later subsections.

AEAP. Automatic E-mail Address Porting (AEAP) [38] allows a user to switch to a new e-mail in Delta Chat and optionally set up forwarding from their old e-mail. When the user sends a message to an existing verified contact from the new address, the contact automatically changes the address for the user with minimal interruption to their chats.

3.1 Autocrypt

Autocrypt [58] is a set of guidelines for e-mail clients to provide low-effort end-to-end e-mail encryption to users by automating key management. The current version, Autocrypt Level 1 (release 1.1.0), provides opportunistic encryption with OpenPGP. It is based on the trust on first use (TOFU) principle and does not support key verification, only aiming to defend against passive data collection attacks. In Autocrypt, an "encrypted" e-mail is always signed and encrypted.

Autocrypt e-mail clients attach their public keys in the `Autocrypt` e-mail header. They may also include `Autocrypt-Gossip` MIME headers in e-mail messages to distribute the public keys of their peers. Autocrypt e-mail clients maintain states for communication peers in a table based on the Autocrypt-related headers in incoming e-mails, and use this to check whether encryption is possible or recommended for a particular e-mail message. The specification also defines the Autocrypt Setup Message to enable key transfer between different e-mail clients.

Delta Chat implements all functionalities in the Autocrypt standard. Other e-mail clients have, if any, only limited support for `Autocrypt` headers. Where relevant, we describe Delta Chat's additions to the standard Autocrypt features.

3.1.1 OpenPGP Keys

We cover a subset of OpenPGP keys that have the same structure as the OpenPGP transferable public key in the Autocrypt standard, which we call Autocrypt-compliant. An Autocrypt-compliant OpenPGP key consists of five parts: (i) a primary key for signing, (ii) a User ID, (iii) a self-signature that certifies the User ID and the primary key, (iv) an encryption subkey, and (v) a signature that binds the subkey using the primary key [58, Section 3.1.1]. While Delta Chat only generates Autocrypt-compliant OpenPGP keys, it actually supports the usage of most valid OpenPGP keys.

The OpenPGP key fingerprint is a 20-byte SHA-1 digest derived from the the public part of the primary key. Other parts of the OpenPGP key do not contribute to the fingerprint.

3.1.2 Autocrypt Header

The `Autocrypt` header is an e-mail header that contains the sender's address `addr` and public key data `keydata`, as well as an optional `prefer-encrypt` attribute that indicates the agreement to perform encryption with peers having the same preference. Autocrypt e-mail clients should include these headers in every message for peers that also use Autocrypt. For this reason, Autocrypt traffic is easily recognisable.

The `keydata` attribute represents an OpenPGP transferable public key (Section 3.1.1). The content of the User ID packet is decorative. The default signing and public-key encryption algorithms in Autocrypt are Ed25519 and ECDH over Curve25519 respectively, and RSA PKCS#1 v1.5 should also be supported. Delta Chat additionally supports NIST P-256 and P-384 curves. See [58, Section 7.2] for an example e-mail with an `Autocrypt` header.

In Autocrypt, the party receiving a message with an Autocrypt header should check whether the `addr` attribute matches the `From` header in the e-mail, rejecting the header but keeping the message if not. Then, it should place the peer's public key in the peer state table (Section 3.1.4) unless there is already an entry for the peer's Autocrypt public key, and the *effective date* (sending time of the message, capped to now) of the message is less recent than that of the existing entry.

3.1.3 Autocrypt Key Gossip

Autocrypt specifies the key gossip mechanism to allow users to send encrypted replies to a group of recipients without having to communicate with each of them in advance. Autocrypt e-mail clients may include `Autocrypt-gossip` headers in the outbound message, one for each of the intended recipients (excluding Bcc). They must be placed in the root MIME part of the encrypted message payload. Each `Autocrypt-gossip` header contains the address and the public key of a recipient. The public key is taken from the peer state table (Section 3.1.4), and for this purpose, Autocrypt prefers the peer's public key (updated with the `Autocrypt` header) over the gossip key in the peer state table. The `Autocrypt-gossip` header has the same format as the `Autocrypt` header, except that the `prefer-encrypt` attribute should not be included. Delta Chat includes the `prefer-encrypt` attribute nonetheless.

Gossip keys in the peer state table are updated in a similar way to Autocrypt keys. Namely, the receiver checks if the address in an `Autocrypt-gossip` header matches some address in the e-mail's `To`, `Cc`, or `Reply-To` header, ignoring the header if not, and updates the peer's gossip key in the peer state table accordingly unless the peer's current gossip key in the table has a more recent *effective date*.

3.1.4 Peer State Table

Autocrypt e-mail clients maintain a state for each peer in a *peer state table*. A peer state records the encryption information of the peer, such as keys and the encryption preference, as well as metadata that can be used by the client to decide on whether to update the state. The client updates the peer states automatically based on the incoming e-mails, relieving the user from OpenPGP key management. When sending a message, the e-mail client uses the peer state table to check whether encryption is possible or preferred, and the client automatically selects the appropriate keys for encryption based on the recipients' states in the peer state table.

Figure 1: Simplified overview of a peer state in Delta Chat.

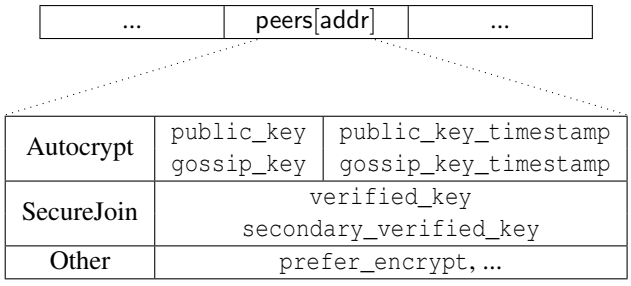

Figure 1 gives an overview of a peer state in Delta Chat, which is augmented with fields for verified keys for the SecureJoin protocols (Section 3.2). Unlike Autocrypt keys, a network attacker should not be able to change verified keys in the peer state table to attacker-controlled keys by modifying network messages. Verified chats in Delta Chat, implemented as verified groups (Section 3.2.3), use exclusively verified keys for chat messages.

Delta Chat also adds a secondary verified key column to the peer state table. This seems to be in place to enable a faster recovery from key changes.[11] When marking a gossip key as verified, if there is already a verified key for the peer and the gossip key is different from the verified key, then Delta Chat sets the peer's secondary verified key to be the gossip key. When receiving an encrypted message from the peer signed with the verified key, Delta Chat removes the secondary verified key for the peer; otherwise, if the message is signed with the secondary verified key, Delta Chat replaces the verified key with the secondary verified key. That is, depending on which key part is actually used by the peer, the two verified keys in the peer state table "resolve" into one.

3.1.5 Autocrypt Setup Message

The Autocrypt specification defines a mechanism to transfer a user's secret key between different Autocrypt e-mail clients via the Autocrypt Setup Message.

The Autocrypt Setup Message is a self-addressed e-mail that contains the user's OpenPGP transferable secret key, ASCII-armored and encrypted with a machine-generated random passphrase called the Setup Code. The user first generates the Autocrypt Setup Message on the old Autocrypt-enabled e-mail client and writes down the Setup Code displayed on screen. Then, the user logs into the new e-mail client with the same e-mail address. The new client finds the Autocrypt Setup Message in the inbox and checks it is well-formed. Then, the new client should ask the user for their approval of the import.[12] If the user agrees, the new client prompts the user to enter the Setup Code for decryption. The new client tries to decrypt the message with the entered Setup Code, and if successful it imports the decrypted secret key.

The Autocrypt Setup Message contains an `Autocrypt-Setup-Message: v1` header but not an `Autocrypt` header. The e-mail body contains instructions in plaintext, and an HTML attachment that contains the user's ASCII-armored and symmetrically-encrypted secret key. Decryption with the correct Setup Code yields the user's ASCII-armored OpenPGP transferable secret key. The Setup Code is used to both decrypt and authenticate the Autocrypt Setup Message. It contains 36 random digits, separated by dashes into blocks of four digits. An example Setup Code is `9503-1923-2307-1980-7833-0983-1998-7562-1111` [58, Section 5.4.2]. The Autocrypt Setup Message may reveal

[11] https://github.com/deltachat/deltachat-core-rust/issues/4541

[12] In Delta Chat, users are not alerted directly but must click on the displayed Setup Message.

the first two digits of the Setup Code. The e-mail client may show these two digits in the prompt as a weak confirmation that the user is entering the correct Setup Code.

3.2 SecureJoin

The SecureJoin protocols [23], namely the Setup Contact protocol and the Verified Group protocol, protect Autocrypt from active network attackers through key verification. They were originally developed as part of the CounterMITM protocols [50], but Delta Chat's implementation has diverged from the original specification in non-trivial ways.[13]

3.2.1 Verification in Delta Chat

A chat being verified indicates that it is a verified group (Section 3.2.3), while a contact being verified indicates that the one-on-one chat with the contact is verified, and the user can add the contact to verified groups. In Delta Chat, key verification is invisible to the user, but it is an important building block for both chat and contact verification.

A contact can become verified by performing the Setup Contact protocol (Section 3.2.2) with the user, or being introduced to a verified group in which the user is a member. Delta Chat marks a contact as verified if (i) it obtains a verified key of the contact, and (ii) it is convinced that the contact is currently using the verified key. For the latter, the contact's verified key should have the same fingerprint as the contact's Autocrypt key in the peer state table, or the gossip key if the Autocrypt key is unavailable. A chat is marked verified upon creation if all chat members are verified. Note that from version 1.133.0, Delta Chat separates contact verification into forward and backward verification (see Appendix B).

3.2.2 Setup Contact Protocol

The Setup Contact protocol allows two peers to exchange e-mail addresses and public keys in a verified manner.

A peer (Alice) first shows her invite code to the other peer (Bob) through a second, out-of-band channel. In Delta Chat, the invite code is usually a QR code, but it can also be a string that the QR code represents. Alice's invite code contains her fingerprint, address, nickname, and two random tokens, INVITE and AUTH, taken from her token database. These tokens do not expire unless Alice manually revokes the corresponding QR code.

After scanning Alice's QR code, Bob's device displays the name and address contained in the QR code, and asks Bob to manually accept the new contact. The rest of the protocol does not require further user interaction, and messages are automatically exchanged using Autocrypt through the channel potentially controlled by the attacker. At the end of the protocol, both participants will have learned and validated the contact information and the public key of their peer. Delta Chat will mark the peer as verified.

A detailed overview of the protocol flow can be found in Fig. 4 in Appendix C, however it is not necessary for the understanding of our attacks.

The SecureJoin document gives an informal argument why the Setup Contact protocol is resistant to impersonation from active attackers that can modify, create and delete messages [23, Section 2.1.2]. However, it acknowledges that an attacker could replay Bob's messages in the protocol to Alice, which could potentially be used to make Alice switch to a compromised key of Bob [23, Section 2.1.3].

3.2.3 Verified Group Protocol

The verified group in Delta Chat is a type of group chat that is secure against network attacks. Members in a verified group are verified to each other, and all messages in a verified group are signed and encrypted using verified keys.

Initialisation. The SecureJoin document does not specify how to create a verified group. In Delta Chat, a user first creates the group locally with a random group identifier. If all members are verified to the creator, the group is automatically marked as verified. The group is invisible to other members until the creator sends the first message in the group. When a group member receives this message, Delta Chat creates the group locally for that member. The group member's client builds the group member list with the `From` and `To` headers, and the group name and identifier can be found in the `Chat-Group-Name` and `Chat-Group-ID` headers. If a `Chat-Verified: 1` header is also present in the signed and encrypted payload, and the message is signed with the sender's verified key, then the group member's client marks all other members of the group as verified, using the gossip keys in the message as verified keys.

Preparing for joining a verified group. Suppose Alice is a member in some verified group, while Bob is not. For Alice to add Bob to the verified group, Alice should have verified Bob's key, e.g. by performing the Setup Contact protocol with Bob, or by being in another verified group with Bob. If this is not the case, Alice can perform a process very similar to the Setup Contact protocol to prepare for adding Bob to the group, which we summarise below.

First, Alice shares the QR code of the verified group with Bob. The QR code contains Alice's fingerprint and address, the group name and identifier, and two random tokens associated with the group, INVITE_g and AUTH_g, taken from Alice's token database. Tokens are associated with their groups, so different groups use separate tokens.

Bob scans the QR code shared by Alice and manually confirms joining the verified group specified in the QR code.

[13] Note that at the time of writing, the new specification for SecureJoin [23] was not complete with respect to the implementation in Delta Chat.

No further user interaction is required for the rest of the process. After that, the peers' clients perform a variant of the Setup Contact protocol. After verifying Bob's key, Alice's client adds Bob to the verified group similar to the process we describe below.

Joining a verified group. Suppose Alice and Bob are verified to each other. For Alice to add Bob to a verified group, Alice's client first adds Bob to the group locally, and then sends a `Chat-Group-Member-Added` message to all members, including Bob. The message contains the verified keys of all members except Alice herself as `Autocrypt-Gossip` headers in the signed and encrypted payload.

On receiving the message, Bob's client creates the group locally and marks peers as verified using the gossip keys. Other group members' clients check that Alice and they are both members of the group (while Bob may or may not be in the group in their views), mark the gossip keys as verified, and add Bob as a group member.

4 Attacks

In this section, we describe our attacks within the Delta Chat threat model outlined in Section 2. Further attacks outside of their threat model can be found in Appendix E.

4.1 Gossip Key Injection

Delta Chat's implementation of the Verified Group protocol allows an attacker to trick the target into marking an attacker-controlled gossip key as verified. This attack does not require a network attacker: the attacker only needs to know the target's public key and send a future-dated e-mail to the target.

Vulnerability. In a verified group, some messages contain one or more `Autocrypt-Gossip` headers in the protected payload. When adding a member to the group, the introducer sends a `Chat-Group-Member-Added` group message that contains the verified keys of all members, including the new member. In addition, Delta Chat periodically attaches the gossips of other members' verified keys to verified group messages, to allow for a faster recovery from key changes. The recipient's client marks the gossiped keys as verified in the peer state table as described in Section 3.2.3.

Since `Autocrypt-Gossip` headers should only be present in the signed and encrypted payload, the attacker cannot add or modify `Autocrypt-Gossip` headers in verified group messages. However, it is possible for the attacker to trick the target's client into marking a different gossip key than that in the message as verified.

When marking a gossip key as verified, Delta Chat iterates over the intersection of addresses in the gossip headers and addresses in the `To` header.[14] For each address, Delta Chat sets its verified key to be the gossip key in the peer state table, or sets a secondary verified key to be the gossip key if a different verified key is already present in the table. The problem is that there is no guarantee that the peer's gossip key in the peer state table actually corresponds to the gossip key in the message, and it is possible that the peer's gossip key does not get updated by the message.

Delta Chat only updates the peer's gossip key in the table if the gossip key in the message is more recent.[15] When comparing the two timestamps, Delta Chat does not cap the send time of the e-mail as required by the Autocrypt standard [58, Section 3.3, 3.6]. Therefore, if the gossip key for some contact in the peer state table has a more recent timestamp than the verified group message with a gossip for the contact, then Delta Chat marks the gossip key in the table as verified. This is potentially different from the contact's real key.

Attack. Suppose Alice is a member of a verified group, while Bob is either a member in the verified group or is about to be added to the verified group by another member Carol with a `Chat-Group-Member-Added` message. Alice and Bob have not communicated in Delta Chat before. Mallory is an attacker outside the group that can send messages to Alice and knows Alice's public key. Mallory does not need to be able to eavesdrop on, spoof or modify network messages, but these capabilities would be helpful in making use of the attack. Figure 2 illustrates the following attack.

First, Mallory sends an e-mail message to Alice, with Bob also in the `To` header of the e-mail. The message includes a malicious gossip key for Bob in its protected payload. The e-mail is dated from the future, which is not expected behaviour but often occurs in practice since client time misconfiguration issues are common. Alice's client parses the message and updates Bob's gossip key in the peer state table accordingly. The malicious gossip key remains in Alice's peer state table, even if Alice deletes the message or blocks Mallory afterwards.

After a short while, suppose Alice receives a message in the verified group with a gossip key for Bob. Because the malicious gossip key for Bob has a more recent timestamp, the verified group message does not update Bob's gossip key in Alice's peer state table. Alice's client then tries to mark the gossip key for Bob as verified, which has the effect of marking the malicious gossip key as verified instead.

If Bob is a new member to be added to the group, then Mallory already controls Bob's verified key in Alice's view. Otherwise, Mallory controls a secondary verified key for Bob, and in order to promote the malicious key to a verified key,

[14] https://github.com/deltachat/deltachat-core-rust/blob/1edd7045bec1f8fa0761e26a975cb9f07596feb8/src/receive_imf.rs#L2479

[15] https://github.com/deltachat/deltachat-core-rust/blob/1edd7045bec1f8fa0761e26a975cb9f07596feb8/src/peerstate.rs#L332

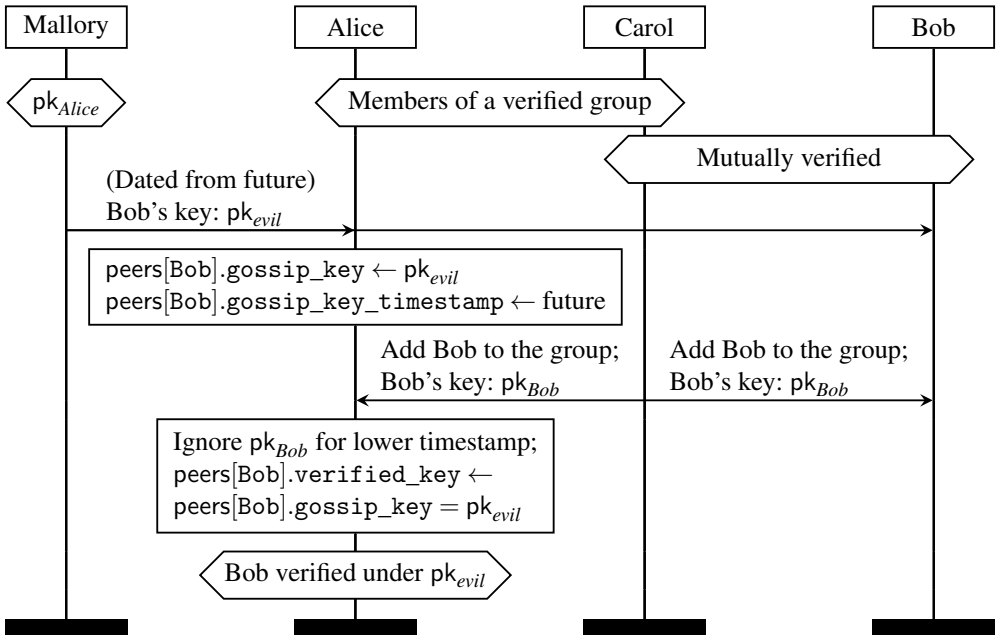

Figure 2: The message sequence for the example gossip key injection attack. Note that the initial message from Mallory to Alice may or may not be actually sent to Bob.

Mallory would need to spoof a message from Bob to Alice that is signed with the malicious gossip key.

Mallory as a network attacker could also perform the attack without sending a future-dated e-mail to Alice. Mallory could intercept a verified group message to Alice that contains gossips, which is easily recognisable from the payload size or from new recipients in the `To` header, and, after sending an e-mail with the malicious gossip key, modify the `Date` header to an earlier time before forwarding the message to Alice.

For Alice to mark Bob as verified, Alice's client also needs to be convinced that Bob is actually using the verified key. If Alice already had Bob's real public key in her peer state table, then Alice's client would not mark Bob as verified, since Bob is potentially using a different key. In that case, to complete the attack, Mallory just needs to also spoof a message from Bob to Alice with the malicious key in the Autocrypt header.

Consequences. By tricking Alice's client to verify a malicious key for Bob, Mallory can eavesdrop on messages from Alice to Bob or in any verified group where Bob is a member. Mallory can also perform a machine-in-the-middle attack between Alice and Bob, despite Alice believing that her communications with Bob are protected with guaranteed end-to-end encryption. Furthermore, Mallory can cause Alice to mark any contact or key as verified by spoofing a message from Bob to Alice which adds Alice to a non-existent verified group with malicious gossips in the message.

Using Delta Chat's AEAP feature (Section 3), Mallory could change Bob's address in Alice's view by sending an e-mail to Alice from the new address. Indeed, Mallory could also use AEAP to make herself verified to Alice, avoiding the need for spoofing e-mails from Bob. However, AEAP would leave a visible user notification to Alice, which could alert her about the attack.

Finally, Alice could unknowingly disseminate the malicious key further in verified groups where Alice is a member, causing more users to verify the malicious key for Bob.

Mitigation. When marking gossiped keys as verified, Delta Chat should not rely on the untrusted gossip key entry in the peer state table. Instead, Delta Chat should directly mark the gossip keys contained in the protected e-mail payload as verified. Further, Delta Chat should use the effective date defined in the Autocrypt standard and cap the message date before updating the peer state table.

4.2 Group Member Removal

By manipulating message headers, it is possible for a network attacker to change a normal group message into the special message for group member removal. As a result, the attacker can remove arbitrary members from a verified group, as if they were removed by some other member in the group.

Vulnerability. Despite the signing and encryption of messages, message headers are surprisingly malleable. By adding

or modifying unprotected e-mail headers of an encrypted message, the attacker can drastically change the meaning and effect of that message. Table 1 lists the manipulatable headers relevant to our attacks. A header in Delta Chat may appear as an IMF header in the unprotected part of the e-mail, or as a MIME header in the possibly signed and encrypted payload. Delta Chat internally categorises headers into four types.

Unprotected: these headers must appear as IMF headers, e.g. `Date` and `Chat-Version`. *Hidden*: these headers can be large and therefore must not appear as IMF headers, e.g. `Chat-User-Avatar`. *Protected*: these headers are encrypted whenever the message is encrypted, e.g. `Chat-Group-Name`. *Secured*: these headers should only be present in the signed and encrypted payload. The `Chat-Verified` and `Secure-Join-Fingerprint` headers are explicitly marked as secured. In addition, Delta Chat treats the `Autocrypt-Gossip` header as secured.

The e-mail parser removes or ignores secured headers that appear in the unencrypted part. However, perhaps counterintuitively, a protected header can appear as an unencrypted IMF header even if the e-mail is signed and encrypted. This design choice is necessary for headers like `Subject` and `From`, which are generally required for well-formed e-mails, but is incorrect for other protected headers, such as `Chat-Group-Member-Removed`, which should only appear in the possibly encrypted e-mail body.

The situation is more complicated when the same protected header appears in both encrypted and unencrypted parts. Delta Chat parses the unencrypted headers before the encrypted headers, preferring a new header over an already parsed one if the header is considered as "known" or starts with `Chat-`. Therefore, the encrypted header generally takes precedence over the unencrypted header. For example, the encrypted `Subject` header is preferred over its unencrypted counterpart, which reflects a common practice in OpenPGP e-mail clients to hide e-mail subjects.

However, because of several oversights in Delta Chat's e-mail parser implementation, there are cases where the unencrypted header could overwrite the encrypted header, including `Secure-Join`, `Secure-Join-Auth` and `Secure-Join-Group`, which are not included in the list of known headers. Moreover, `Secure-Join-Auth` should have been treated as secured instead of protected, as it never appears unencrypted in honest executions.

In addition, the `Message-ID` header and the `From` header are in effect susceptible to overwriting. The `Message-ID` header, while not susceptible to overwriting per se, can be overwritten by the unprotected `X-Microsoft-Original-Message-ID` header, which was used in older versions of Delta Chat and remains for compatibility. For the `From` header, Delta Chat decided not to reject an e-mail whose encrypted `From` header is different from its unencrypted `From` header; however, Delta Chat considers such an e-mail as having an unsigned `From`, meaning that it cannot be used to initiate the AEAP mechanism.

Attack. Suppose Alice and Bob are in the same verified group. If Alice wished to remove Bob from the group, Alice would send a message with a `Chat-Group-Member-Removed` header containing Bob's e-mail address to the members of the verified group. Upon seeing the message, all members' clients, including Bob's, would remove Bob from the verified group, and display a notification saying that Bob has been removed from the group by Alice.

Suppose Mallory is an eavesdropping attacker outside the group that is also capable of spoofing messages to Alice; Mallory can also be a network attacker. First, Mallory captures a non-administrative message that Alice has sent to the verified group at some point in the past. Mallory should be able to capture such a message with reasonable probability, and she can use some characteristics of the message, such as size and the `To` header, to help distinguish it from administrative messages. Mallory can easily distinguish messages in different groups, since the group ID is a part of the plaintext `Message-ID` header.

Then, Mallory modifies the captured message by adding a `Chat-Group-Member-Removed` header with Bob's address to the unprotected IMF header section of the message. Mallory also modifies the unprotected `Message-ID` header so that the message will not be treated as a duplicate by Alice. Newer versions of Delta Chat place a copy of the `Message-ID` header in the protected payload, but the attacker can instead set the unprotected `X-Microsoft-Original-Message-ID` header, which takes precedence over `Message-ID`. (If Mallory could intercept and modify a non-administrative message from Alice to the verified group before it is delivered to the group, she could instead directly add the `Chat-Group-Member-Removed` header with Bob's address to its IMF headers.)

After that, Mallory sends the modified message to all members of the verified group. The members' clients incorrectly interpret the modified message as a group member removal message initiated by Alice. Mallory may choose to send this message to Alice too; her client would also remove Bob upon receiving the message.

Consequences. While the attacker cannot read messages in the verified group, they are able to arbitrarily remove members of the group, which could create confusion about the group's membership state or dissolve the group.

There could be other potential attacks that leverage header manipulation. For example, it is possible to add or remove the avatar of a verified group by adding a `Chat-Group-Avatar` header to a normal group message. It is also possible to change the group name for two peers in a verified group by reusing another message from their one-on-one chat, where other members could not see the change.

Table 1: The types and susceptibility to overwriting of the manipulatable headers in Delta Chat that are relevant to our attacks.

Header	Type	Overwriting	Reference
`Chat-Group-Avatar`	hidden	no	Section 4.2
`Chat-Group-Member-Removed`	protected	no	Section 4.2
`From`	protected	yes	Appendix E
`Message-ID`	protected	yes	Section 4.2
`Secure-Join`	protected	yes	Section 4.4
`Secure-Join-Auth`	protected	yes	Appendix E

Mitigation. An immediate fix to the attack would disallow headers starting with `Chat-` to appear in the unencrypted part if the message is encrypted. However, it takes more careful checks to completely eliminate such attacks. In general, if a protected header appears in the plaintext part of an encrypted message, then Delta Chat should regard the message as invalid. There are a few exceptions for headers generally required for a well-formed e-mail message, such as `Subject` and `From`, in which case Delta Chat should prefer the encrypted header, and only reject the message if the encrypted header is meant to match the unencrypted header but does not.

4.3 Synchronisation Forgery

Delta Chat verifies self-addressed messages with the user's own Autocrypt public key in the peer state table. Consequently, an attacker capable of spoofing self-addressed e-mails could trivially forge self-addressed messages for the target, including synchronisation messages.

Vulnerability. Delta Chat synchronises certain changes in the account across devices. These include the addition or deletion of QR invite code tokens for the SecureJoin protocols, changes to chat statuses like blocking contacts or archiving chats, and changes to a limited set of account configurations like display names. Synchronisation is enabled by default, even if the user only uses Delta Chat on one device.

Delta Chat synchronises these changes by sending a synchronisation message, which is a self-addressed e-mail with a `multi-device-sync.json` file attached. The attachment contains the synchronisation items represented in JSON. Like other messages, the synchronisation message should be signed and encrypted. While Delta Chat encrypts the message correctly with the user's real public key, the verification of a self-addressed e-mail message uses the user's Autocrypt public key as it appears in the peer state table, similarly as for other opportunistically protected one-on-one chats. As with other contacts, Delta Chat updates the user's own Autocrypt key in the peer state table using the Autocrypt headers from the received self-addressed messages.

Attack. An attacker in the possession of the target's public key and capable of spoofing self-addressed e-mails for the target could make the target's client execute malicious synchronisation items crafted by the attacker.

To mount this attack, the attacker creates a message containing the malicious synchronisation items, signed with a key known by the attacker and encrypted with the target's public key. This message contains an Autocrypt header that corresponds to the public part of the signing key. The attacker then sends the message to the target, spoofed to appear as if it was sent by the target.

On receiving the message, the target's Delta Chat client first updates the target's own Autocrypt key entry in the peer state table to the public key in the Autocrypt header, then decrypts the message with the target's own private key, and finally verifies the message with the target's own Autocrypt key in the peer state table, which succeeds since the message is indeed signed with the private counterpart of the Autocrypt key; see Fig. 3. The target's client now processes the message as a valid synchronisation message and proceeds to execute the malicious synchronisation items in the message.

Consequences. By planting AUTH and INVITE tokens in the synchronisation items to the target, the attacker could verify themselves to the target, or impersonate other contacts to the target in the Setup Contact protocol or the Verified Group protocol. The recent Delta Chat updates on synchronisation[16] equip the attacker with more capabilities. The attacker could now change chat statuses on the target's device, such as blocking contacts or muting chats, and force the target to change several configurations, such as the display name. Note that in most cases, executing synchronisation items does not result in a notification to the user.

More generally, the attacker can forge self-addressed messages with similar techniques, which appear to be encrypted in the "Saved Messages" chat on the victim's client. While this chat does not bear a green checkmark, the target (e.g. a Delta Chat bot[17]) may still perceive it as verified and incorrectly

[16] See e.g. https://github.com/deltachat/deltachat-core-rust/pull/4843 and https://github.com/deltachat/deltachat-core-rust/pull/5023.

[17] https://bots.delta.chat/

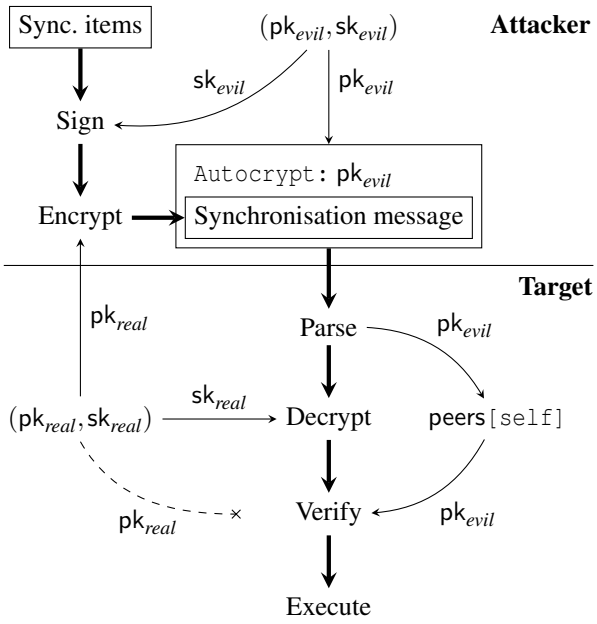

Figure 3: Keys involved in the synchronisation forgery attack, where $(\mathsf{pk}_{real}, \mathsf{sk}_{real})$ is the target's key pair, and $(\mathsf{pk}_{evil}, \mathsf{sk}_{evil})$ is a key pair known by the attacker. Ideally, Delta Chat should use pk_{real} to verify the synchronisation message (dashed line).

regard the forged message as self-addressed.

Mitigation. Delta Chat should handle self-addressed messages separately from other messages, and in particular only use the user's own key to verify self-addressed messages.

Device synchronisation could be made more secure by notifying the user about the changes that were executed, and using timestamps to prevent synchronisation replays.

4.4 InsecureJoin Observer

We describe another attack that leverages header manipulation (Section 4.2) to trick Delta Chat into marking an attacker-controlled key as verified. The attack relies on a mechanism in Delta Chat that observes the SecureJoin protocol instance on other devices.

Vulnerability. When Delta Chat sees a self-sent SecureJoin message not originating from the current device to a peer, it interprets it as the user establishing contact with the peer on another device. If the message is correctly signed and encrypted, then Delta Chat tries to use the SecureJoin message in order to achieve some form of synchronisation between devices. Further, if the message indicates the final stages of the protocol, i.e. it can be deduced that the other device has verified or will shortly be able to verify the peer, then Delta Chat also marks the gossip key for the peer as verified on this device.

Similar to the group member removal attack (Section 4.2), the vulnerability we exploit in this attack is the lack of protection for headers in Delta Chat, and more specifically, the flexibility of the `Secure-Join` header. Delta Chat uses the `Secure-Join` header to indicate a SecureJoin message and denote its type (e.g. `vc-request`). The `Secure-Join` header is treated as protected, meaning that it should be encrypted whenever the message is encrypted. However, this header is incorrectly allowed to appear unencrypted even when the message is encrypted. Moreover, an unencrypted `Secure-Join` header overwrites an encrypted `Secure-Join` header in the same message.

In addition to the attack we describe here, a variant of the gossip key injection attack (Section 4.1) still seems applicable in this context, because when marking a peer's gossip key as verified based on the observed SecureJoin message, Delta Chat would look for the gossip key in the peer state table rather than in the protected message payload. Our attack does not exploit this vulnerability. An attacker cannot use the technique in the synchronisation forgery attack (Section 4.3) to forge a self-sent SecureJoin message, because Delta Chat would check that the message is signed with the user's real key.

Attack. Suppose the attacker is able to spoof messages sent from the target. In addition, we assume the attacker is able to eavesdrop on the communications of the target; alternatively, we could instead assume that the target replies to messages sent from the attacker. To make the target verify an attacker-controlled key, the attacker crafts a message perceived by the target's Delta Chat client as a valid SecureJoin message from the target to the attacker or to an address of the attacker's choice, which actually contains an attacker-controlled key as a gossip header.

The general idea of crafting such a message is to first trick the target into signing a message that contains the attacker-controlled key as a gossip key, and then add a `Secure-Join` header indicating a SecureJoin message in a final protocol stage (e.g. `vc-contact-confirm`) to the unprotected IMF headers of the message.

There are several possible ways of making the target sign the attacker-controlled gossip key. For example, the attacker could send messages to the target in unverified one-on-one chats or groups, hoping the target would reply. The first reply would carry a gossip of the attacker's public key, and gossips would also occur in replies periodically. The attacker could also send a `vc-request` message to the target. The `vc-request` message is a cleartext message in the Setup Contact protocol that carries an INVITE token in the target's QR invite code; the attacker can eavesdrop on past Setup Contact protocol instances to learn a valid INVITE for the target. The target replies with a `vc-auth-required` message, which contains a gossip of the attacker's public key in the protected payload. Because the unprotected `Secure-Join` header takes precedence, adding an unprotected `Secure-Join` header to

the target's reply can change the SecureJoin message to a desired type.

The attacker then sends the crafted message to the target via Bcc as if the target sent it from another device. The target's Delta Chat client would recognize the message as a valid SecureJoin message on another device that indicates final stages of the protocol, and mark the attacker's gossip key in the message as verified.

The changes to Delta Chat core that introduce forward and backward verification would only impact the attack slightly – the attacker would need to send one more message to the target (see Appendix B).

Consequences. The consequences for this attack largely overlap with the gossip key injection attack (Section 4.1). However, this attack is harder to mount, as it requires stronger capabilities for the attacker.

Mitigation. As this attack stems from header flexibility, fixing the header manipulation vulnerability, and in particular, disallowing the `Secure-Join` header to appear unencrypted whenever the message is encrypted, would prevent the attack.

4.5 Autocrypt Setup Forgery

The Autocrypt Setup Message transfers the user's encrypted secret key over an insecure channel, which could make it vulnerable to key overwriting attacks [17]. However, the Autocrypt Setup Message includes the user's public key, which prevents classical attacks of this type. Despite this, because the rPGP library has a bug that allows decryption using the "plaintext" algorithm, it is easy to forge a valid Autocrypt Setup Message under an unknown Setup Code.

Vulnerability. The Setup Code plays an important role in guaranteeing the security of the Autocrypt Setup Message. To quote the Autocrypt specification: *"The Code serves both for decryption as well as authenticating the message. Extra care needs to be taken with some PGP implementations that the Setup Code is actually used for decryption. For example, this is difficult to do correctly with GnuPG."* [58, Section 5.4.4]

While RFC 4880 specifies "plaintext" as a symmetric-key algorithm with ID 0, the specification also stresses that the plaintext algorithm *"may only be used to denote secret keys that are stored in the clear"*, and that *"Implementations MUST NOT use plaintext in Symmetrically Encrypted Data packets"* [35, Section 13.4]. However, it is possible in rPGP to use the plaintext algorithm to "decrypt" session keys or data. In rPGP, the plaintext algorithm has a "key size" and a "block size" of 0. There are no checks on the key size for the plaintext algorithm. The algorithm simply strips the first two bytes of the data, which it perceives as the random block and "quick check" bytes, and returns the rest.

Attack. The attack idea is straightforward: a network attacker can substitute the encrypted data in the target's Autocrypt Setup Message with a secret key controlled by the attacker, encrypted under the plaintext algorithm. The malicious message "decrypts" correctly to the attacker-controlled key under the original Setup Code. This breaks the expectation of the Autocrypt standard which claims that Setup Codes always authenticate the message. In the end, the target's client imports and uses a key controlled by the attacker.

Alternatively, if the attacker was only able to spoof self-addressed messages for the target, the attacker could still send the malicious message along with the honest one, hoping the target would click on the malicious message. As a last resort, since Delta Chat allows Setup Messages that are not self-addressed, the attacker could simply send the malicious message to the target in an unverified chat, possibly changing the avatar and name to resemble the chat "Saved Messages" so as to confuse the target. Details of how to construct the OpenPGP packets required to exploit the plaintext algorithm can be found in Appendix D.

Consequences. The attacker can make the target accept a malicious Autocrypt Setup Message and start to use a key controlled by the attacker. The attacker is then able to read all of the target's future communications as well as impersonate the target to others after the target verifies the malicious key to them. However, since the malicious key does not contain the secret key materials of the real key, if the target did not stop using their old Autocrypt e-mail client or Delta Chat client, they would soon notice a discrepancy in the messages seen on different devices.

Mitigation. The rPGP library should disable the plaintext algorithm, and Delta Chat should update to the patched version of rPGP. In addition, Delta Chat should not allow Autocrypt Setup Messages that are not self-addressed, and should consider warning users about possible phishing attacks related to such messages.

A Delta Chat user could also manually export and import the secret key or the whole backup, both unencrypted, or use an experimental Delta Chat feature to transfer encrypted account data between two devices within the same network.

4.6 Compression Quine

We describe a known attack on OpenPGP libraries that we found applicable to rPGP. The rPGP library does not have a maximum recursion depth for OpenPGP messages, so it is possible to perform a denial-of-service attack on rPGP, and consequently Delta Chat.

Vulnerability. First publicly identified by Campbell [19], it is known that some OpenPGP implementations could be

vulnerable to a denial-of-service attack that leverages an OpenPGP *compression quine*, which is an OpenPGP Compressed packet that decompresses to itself. Trying to decompress the compression quine may cause the program to exhaust memory and crash, and, in the worst case, dump secret information upon crashing. This appears as an inherent issue in the OpenPGP standard that was not considered by Delta Chat. The draft on OpenPGP "crypto refresh" lists this attack as a security consideration for OpenPGP implementors [62, Section 13.14].

Attack. The attack is straightforward: an attacker with the knowledge of the target's address and public key simply sends an OpenPGP compression quine to the target, encrypted under the target's public key; there is no need for the target to recognize or manually accept the contact.

Consequences. Delta Chat crashes almost immediately upon receiving the message, and the issue persists as long as Delta Chat is connected to the e-mail server. To recover from the attack, the target must manually delete the message from the server, switch to different keys or accounts while offline, or reinstall Delta Chat.

Mitigation. The rPGP library should set a maximum recursion depth for OpenPGP messages. It is also a good idea for rPGP to check against some of the tests in the OpenPGP interoperability test suite.[18] However, without further defenses, the attacker would still able to send a compressed message with an exorbitant compression ratio, also known as a "ZIP bomb", to achieve a similar effect as a compression quine. Therefore, rPGP could consider treating Compressed packets that exceed a maximum compression ratio or contain data that is too large as invalid.

5 Discussion

While it is difficult to estimate the size of Delta Chat's user base, its promotion to users in high-risk contexts makes it a potential target for powerful real-life adversaries. We have shown that attacks are possible under the threat model envisioned by Delta Chat's developers, raising several questions.

Cause of the attacks. Most of the attacks we found leverage deviations from the OpenPGP, Autocrypt, or SecureJoin specifications, rather than attack the protocols or standards themselves. However, the specifications can be unclear or even misleading in some cases, and attacks stemming from cross-protocol interactions can be notoriously difficult to prevent without a comprehensive understanding of the entire system. We believe that some of the attacks could have been found earlier if an attempt to prove the security of the SecureJoin protocols had been made.

The SecureJoin protocols "consider usability, cryptographic and implementation aspects simultaneously, because they constrain and complement each other" [23]. However, balancing security and usability is a hard problem; for users in high-risk contexts, prioritising feature development may increase harm. For example, we found several vulnerabilities related to Delta Chat's multi-device support. Other decentralised messaging applications used in high-risk contexts like Briar[19] do not support multiple devices, thereby limiting their attack surface. Yet, without certain features, users may switch to other applications; indeed, multi-device support was identified as a necessary feature in Delta Chat's interviews with high-risk users [32].

Appropriateness of Delta Chat's threat model. While our attacks fall within the threat model outlined by Delta Chat, it is also important to assess whether this threat model is appropriate in the first place. Prior work [37] has questioned whether opportunistic encryption in the form of Autocrypt should be promoted to users who may not have a full understanding of the implications of its use; on the other hand, Delta Chat has repeatedly incorporated feedback from groups of high-risk users into its development process [32–34], an undertaking that is uncommon but vital for a secure messaging application. It is also useful to contrast Delta Chat's explicit approach to verification with that of Signal, in which users who do not choose the option of verifying safety numbers are participating in *de facto* unverified chats.

The threat model of verified chats, however, omits several categories of attacks such as attacks on privacy, insider attacks, but also attacks that are prevented by other messaging applications as a matter of course. We give a detailed overview of attacks outside of Delta Chat's threat model in Appendix E.

Securing decentralised communications. What could be done to make Delta Chat, or decentralised messaging services more widely, more secure for the future? We argue that more analysis is necessary on multiple levels.

Commercial security audits can increase trust in a particular product by helping to eliminate implementation flaws and insecure configurations. Delta Chat's two cryptographic audits showed overall positive results, having found only two "high-risk" issues. However, the scope of such audits is, by definition, limited. Our results demonstrate that passing a security audit cannot be seen as a security end-goal in itself.

However, neither can analysis such as ours be seen as a definitive stamp of approval. Beyond immediate mitigation, questions arise about the future of open specifications such as Autocrypt and SecureJoin. Formal cryptographic proofs such as [5] can help in gaining a better understanding of the security

[18] https://tests.sequoia-pgp.org/

[19] https://briarproject.org/

guarantees given by a particular design; yet, in cases such as Delta Chat, it can be challenging to make formal statements about a design that lacks a protocol specification and whose cryptographic implementation changes frequently. The work on ClaimChain [44] in particular shows that even if a design with stronger security properties exists, circumstances may prevent it being adopted in practice.

We do not intend to contribute to the already abundant discussion on whether OpenPGP should be considered insecure by modern standards. However, our attacks show that one should take great caution when using OpenPGP in non-standard scenarios as well as when validating OpenPGP messages. We believe Delta Chat should limit its usage of OpenPGP to a more secure subset, and consider upgrading to crypto-refresh [62] in the near future.

Instead of further customising the SecureJoin protocols, pushing for and contributing to a specification of Autocrypt Level 2 could provide a way forward for Delta Chat. This was alluded to in the current Autocrypt specification [58] as providing protection against active attackers but work on it never began. However, such an undertaking would require participation from the wider community. In the meantime, the lack of viable alternative tools for secure decentralised messaging makes it a pressing problem for developers and researchers alike.

References

[1] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 5–17. ACM Press, October 2015.

[2] Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková. Collective information security in large-scale urban protests: the case of Hong Kong. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3363–3380. USENIX Association, August 2021.

[3] Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková. Mesh messaging in large-scale protests: Breaking Bridgefy. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 375–398. Springer, Heidelberg, May 2021.

[4] Martin R. Albrecht, Sofía Celi, Benjamin Dowling, and Daniel Jones. Practically-exploitable cryptographic vulnerabilities in Matrix. In *2023 IEEE Symposium on Security and Privacy (S&P)*, pages 164–181, 2023.

[5] Martin R. Albrecht, Benjamin Dowling, and Daniel Jones. Device-oriented group messaging: A formal cryptographic analysis of Matrix' core. 2023. https://eprint.iacr.org/2023/1300.

[6] Martin R. Albrecht, Raphael Eikenberg, and Kenneth G. Paterson. Breaking Bridgefy, again: Adopting libsignal is not enough. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 269–286, Boston, MA, August 2022. USENIX Association.

[7] Martin R. Albrecht, Lenka Mareková, Kenneth G. Paterson, and Igors Stepanovs. Four attacks and a proof for Telegram. In IEEE S&P 2022 [40], pages 87–106.

[8] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540. IEEE Computer Society Press, May 2013.

[9] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2019.

[10] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Heidelberg, August 2020.

[11] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Modular design of secure group messaging protocols and the security of MLS. In Vigna and Shi [60], pages 1463–1483.

[12] Joël Alwen, Marta Mularczyk, and Yiannis Tselekounis. Fork-resilient continuous group key agreement. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 396–429. Springer, Heidelberg, August 2023.

[13] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. DROWN: Breaking TLS using SSLv2. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 689–706. USENIX Association, August 2016.

[14] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon.

The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023.

[15] Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. A more complete analysis of the Signal double ratchet algorithm. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 784–813. Springer, Heidelberg, August 2022.

[16] Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the station-to-station (STS) protocol. In Hideki Imai and Yuliang Zheng, editors, *PKC'99*, volume 1560 of *LNCS*, pages 154–170. Springer, Heidelberg, March 1999.

[17] Lara Bruseghini, Daniel Huigens, and Kenneth G. Paterson. Victory by KO: Attacking OpenPGP using key overwriting. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 411–423. ACM Press, November 2022.

[18] Chris Brzuska, Eric Cornelissen, and Konrad Kohbrok. Security analysis of the MLS key derivation. In IEEE S&P 2022 [40], pages 2535–2553.

[19] Taylor R. Campbell. On compression in data formats. https://web.archive.org/web/20181116185051/https://mumble.net/~campbell/2013/10/08/compression.

[20] Delta Chat. FAQ - Delta Chat. https://web.archive.org/web/20240206093339/https://delta.chat/en/help.

[21] Delta Chat. Hardening guaranteed end-to-end encryption based on a security analysis from ETH researchers. https://delta.chat/en/2024-03-25-crypto-analysis-securejoin.

[22] Delta Chat. User stories and voices. https://web.archive.org/web/20240206093211/https://delta.chat/en/user-voices.

[23] Delta Chat. SecureJoin: Protecting chat messaging against network adversaries. https://web.archive.org/web/20240110171400/https://securejoin.readthedocs.io/en/latest/, 2023.

[24] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the Signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983, October 2020.

[25] Cas Cremers, Charlie Jacomme, and Aurora Naska. Formal analysis of session-handling in secure messaging: Lifting security from sessions to conversations. Cryptology ePrint Archive, Report 2022/1710, 2022. https://eprint.iacr.org/2022/1710.

[26] Luca De Feo, Bertram Poettering, and Alessandro Sorniotti. On the (in)security of ElGamal in OpenPGP. In Vigna and Shi [60], pages 2066–2080.

[27] Front Line Defenders. Guide to secure group chat and conferencing tools. https://web.archive.org/web/20230922042917/https://www.frontlinedefenders.org/en/resource-publication/guide-secure-group-chat-and-conferencing-tools.

[28] L. Peter Deutsch and Jean loup Gailly. ZLIB compressed data format specification version 3.3. RFC 1950, 1996.

[29] Christoph Döberl, Wolfgang Eibner, Simon Gärtner, Manuela Kos, Florian Kutschera, and Sebastian Ramacher. Quantum-resistant end-to-end secure messaging and email communication. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ARES '23, New York, NY, USA, 2023. Association for Computing Machinery.

[30] Benjamin Dowling, Felix Günther, and Alexandre Poirrier. Continuous authentication in secure messaging. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part II*, volume 13555 of *LNCS*, pages 361–381. Springer, Heidelberg, September 2022.

[31] Benjamin Dowling and Britta Hale. Secure messaging authentication against active man-in-the-middle attacks. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 54–70, 2021.

[32] Ksenia Ermoshina. Needfinding report – Multitool contexts and organizational features. https://web.archive.org/web/20230521003456/https://delta.chat/en/2020-03-31-needfinding_multidevice, 2020.

[33] Ksenia Ermoshina and Vadym Hudyma. Needfinding report based on interviews in Ukraine. https://web.archive.org/web/20231128092711/https://delta.chat/en/2018-12-19-needfinding, 2018.

[34] Ksenia Ermoshina and Vadym Hudyma. Delta chat UX and needfinding final report. https://web.archive.org/web/20221225004507/https://delta.chat/assets/blog/Delta-Chat-UX-final-report-july2019.pdf, 2019.

[35] Hal Finney, Lutz Donnerhacke, Jon Callas, Rodney L. Thayer, and Daphne Shaw. OpenPGP message format. RFC 4880, 2007.

[36] Yoel Gluck, Neal Harris, and Angelo Prado. BREACH: Reviving the CRIME attack. *Black Hat*, 2013.

[37] Harry Halpin. SoK: why Johnny can't fix PGP standardization. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ARES '20, New York, NY, USA, 2020. Association for Computing Machinery.

[38] hocuri and holga. Introducing Automatic E-mail Address Porting (AEAP). https://web.archive.org/web/20231128100514/https://delta.chat/en/2022-09-14-aeap.

[39] holga. Guaranteed End-to-End encryption and many other good news. https://web.archive.org/web/20240111214108/https://delta.chat/en/2023-11-23-jumbo-42.

[40] *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2022.

[41] Fabian Ising, Damian Poddebniak, Tobias Kappert, Christoph Saatjohann, and Sebastian Schinzel. Content-Type: multipart/oracle - tapping into format oracles in email End-to-End encryption. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4175–4192, Anaheim, CA, August 2023. USENIX Association.

[42] John Kelsey. Compression and information leakage of plaintext. In *FSE*, volume 2365 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2002.

[43] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 429–448. Springer, Heidelberg, August 2013.

[44] Bogdan Kulynych, Wouter Lueks, Marios Isaakidis, George Danezis, and Carmela Troncoso. ClaimChain: Improving the security and privacy of in-band key distribution for messaging. In *WPES@CCS*, pages 86–103. ACM, 2018.

[45] Timo Longin and SEC Consult. SMTP smuggling - spoofing e-mails worldwide. https://web.archive.org/web/20240123001533/https://sec-consult.com/blog/detail/smtp-smuggling-spoofing-e-mails-worldwide/.

[46] Florian Maury, Jean-René Reinhard, Olivier Levillain, and Henri Gilbert. Format oracles on OpenPGP. In *CT-RSA*, volume 9048 of *Lecture Notes in Computer Science*, pages 220–236. Springer, 2015.

[47] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. A modular security analysis of the TLS handshake protocol. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 55–73. Springer, Heidelberg, December 2008.

[48] Jens Müller, Marcus Brinkmann, Damian Poddebniak, Hanno Böck, Sebastian Schinzel, Juraj Somorovsky, and Jörg Schwenk. "Johnny, you are fired!" – spoofing OpenPGP and S/MIME signatures in emails. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1011–1028, Santa Clara, CA, August 2019. USENIX Association.

[49] NEXTLEAP. NEXTLEAP project. https://web.archive.org/web/20240111234949/https://nextleap.eu/, 2018.

[50] NEXTLEAP. Detecting and preventing active attacks against Autocrypt. https://web.archive.org/web/20231208095502/https://countermitm.readthedocs.io/en/latest/, 2020.

[51] Kenneth G. Paterson, Matteo Scarlata, and Kien Tuong Truong. Three lessons from Threema: Analysis of a secure messenger. In *USENIX Security Symposium*, pages 1289–1306. USENIX Association, 2023.

[52] Damian Poddebniak, Christian Dresen, Jens Müller, Fabian Ising, Sebastian Schinzel, Simon Friedberger, Juraj Somorovsky, and Jörg Schwenk. Efail: Breaking S/MIME and OpenPGP email encryption using exfiltration channels. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 549–566. USENIX Association, August 2018.

[53] The dComms Project. dComms: Alternative communication methods during censorship, interruptions, and restrictions on the Internet in Ukraine. https://web.archive.org/web/20240206093610/https://dcomm.net.ua/en/.

[54] Juliano Rizzo and Thai Duong. The CRIME attack. *Ekoparty*, 2012.

[55] Scott Ruoti, Jeff Andersen, Daniel Zappala, and Kent Seamons. Why Johnny still, still can't encrypt: Evaluating the usability of a modern PGP client. https://arxiv.org/abs/1510.08555, 2016.

[56] Include Security. Security assessment of DeltaChat's RPGP and RustCrypto RSA libraries for the Open Tech Fund, 2019. https://web.archive.org/web/20230605134243/https://delta.chat/assets/blog/2019-first-security-review.pdf.

[57] Include Security. Security assessment of Delta Chat's primary Rust libraries on behalf of the Open Technology Fund, 2020. https://web.archive.org/web/20230605132024/https://delta.chat/assets/blog/2020-second-security-review.pdf.

[58] Autocrypt team. Autocrypt Level 1 specification. https://web.archive.org/web/20231207173301/https://autocrypt.org/autocrypt-spec-1.1.0.pdf, 2020.

[59] Elham Vaziripour, Justin Wu, Mark O'Neill, Daniel Metro, Josh Cockrell, Timothy Moffett, Jordan Whitehead, Nick Bonner, Kent Seamons, and Daniel Zappala. Action needed! helping users find and complete the authentication ceremony in Signal. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 47–62, Baltimore, MD, August 2018. USENIX Association.

[60] Giovanni Vigna and Elaine Shi, editors. *ACM CCS 2021*. ACM Press, November 2021.

[61] Alma Whitten and J. Doug Tygar. Why johnny can't encrypt: A usability evaluation of PGP 5.0. In G. Winfield Treese, editor, *USENIX Security 99*. USENIX Association, August 1999.

[62] Paul Wouters, Daniel Huigens, Justus Winter, and Niibe Yutaka. OpenPGP draft-ietf-openpgp-crypto-refresh-13. https://datatracker.ietf.org/doc/draft-ietf-openpgp-crypto-refresh/13/, 2024.

A OpenPGP

OpenPGP [35] is a standard for providing confidentiality and authenticity for messages and data, primarily used for end-to-end e-mail encryption. The main standard for OpenPGP is RFC 4880 [35]. In this section, we summarise only the parts of OpenPGP that are relevant for our analysis.

Packets. In OpenPGP, messages and keys consist of one or more records called *packets*, and some of the packets may in turn contain nested packets. Each packet consists of a variable-length packet header, which specifies the type and length of the packet, and the packet body, which contains the actual data. Table 2 lists the types of packets that are most important for our analysis of Delta Chat.

Signatures. OpenPGP uses digital signatures for a wide range of purposes: the OpenPGP standard lists 15 different signature types [35, Section 5.2.1]. Signatures on binary or text messages attest to their authenticity, and signatures on public keys can bind attributes and subkeys to the primary key or certify the owner's identity.

Table 2: Selected packet types in OpenPGP [35, Section 4.3].

Packet Type	**Tag**
Public-Key Encrypted Session Key (PKESK)	1
Signature	2
Symmetric-Key Encrypted Session Key (SKESK)	3
Public-Key	6
Compressed Data	8
Symmetrically Encrypted and Integrity Protected Data (SEIPD)	18

Digital signatures in OpenPGP are represented with the *Signature packet* [35, Section 5.2]. A Signature packet contains a signature on some data as well as information about the signature, such as the signature type and the public-key algorithm used. Delta Chat uses version 4 Signature packets by default, and version 3 Signature packets are also accepted. Table 3 shows the body of a version 4 Signature packet.

To compute a signature, an OpenPGP implementation first computes a hash over the data to be signed; more precisely, the hash digest covers the data being signed and a prefix of the Signature packet body, inclusive from the version number to hashed subpackets. Then, it uses the specified signature algorithm to compute the signature over the hash under the user's private key. Similarly, to verify a signature under a public key, an implementation first computes the hash from the data to be verified, and then uses the signature algorithm specified in the Signature packet to verify the signature under the corresponding public key.

Compression. While it is possible to encrypt uncompressed messages in OpenPGP, it is usually the case that the message is compressed after signing and before encryption. The default compression format in Delta Chat is zlib [28]. OpenPGP represents compressed data in a *Compressed Data packet* [35, Section 5.6].

RFC 4880 requires that "decompressing a Compressed Data packet must yield a valid OpenPGP Message" [35, Section 11.3]. However, an OpenPGP message can come in many different shapes and forms; in particular, a single Compressed Data packet already qualifies as a valid OpenPGP message.

Encryption. We describe a typical workflow in OpenPGP for message encryption [35, Section 2.1]. First, the OpenPGP implementation generates a random session key, and uses the session key to encrypt the message as well as a SHA-1 digest of the message, obtaining a *Symmetrically Encrypted and Integrity Protected Data (SEIPD) packet*. Then, for each intended recipient, the OpenPGP implementation encrypts the session key in an *Encrypted Session Key (ESK) packet*, under the public key of that recipient or under a shared passphrase.

Table 3: The body of a version 4 Signature packet.

version (1B)	sig type (1B)	pubkey alg (1B)	hash alg (1B)
hashed subpackets length (2B)		hashed subpackets (variable)	
unhashed subpackets length (2B)		unhashed subpackets (variable)	
left 16 bits of hash (2B)		signature (variable)	

Table 4: The body of a version 4 SKESK packet.

version (1B)	symkey alg (1B)	S2K specifier (variable)	esk (variable, optional)

The encrypted message is the concatenation of the ESK packets and the SEIPD packet.

On decryption, the OpenPGP implementation first decrypts the corresponding Session Key packet with the private key or the shared passphrase. Then, it decrypts the SEIPD packet using the decrypted session key and checks that the decrypted SHA-1 digest is correct, returning an error if not.

SEIPD packet. The *Symmetrically Encrypted and Integrity Protected Data (SEIPD) packet* contains symmetrically-encrypted data that is integrity protected using the *Modification Detecton Code (MDC)*. To derive an SEIPD packet, an OpenPGP implementation appends an MDC packet to the plaintext data to encrypt, where the MDC packet body is a 20-byte SHA-1 hash of all bytes preceding the packet body, including the plaintext data and the fixed 2-byte MDC packet header. The SEIPD packet body is then the encyption of the plaintext data and the MDC packet under the specified symmetric-key algorithm.

Upon decryption, the MDC is recalculated, and any difference is considered a failure and should be reported to the user [35, Section 5.12]. However, some OpenPGP implementations may only display a warning for MDC failures, making the MDC largely ineffective. The rPGP library correctly treats the MDC failure as an error.

Note that the rPGP library does not support the *Symmetrically Encrypted Data (SED) packet*, which has no integrity protection and is generally considered insecure.

The default OpenPGP symmetric-key algorithm in Delta Chat is AES-128. OpenPGP uses symmetric-key algorithms in the cipher feedback (CFB) mode. In OpenPGP, the IV is of all zeros and is not included in the ciphertext. Instead, OpenPGP samples the first plaintext block at random, and sets the first two bytes in the second plaintext block the same as the last two bytes in the first block. The actual data starts from the third byte of the second plaintext block. These two bytes can act as a "quick check" to confirm that the session key is correct, but performing the quick check may create a format oracle for chosen ciphertext attacks [35, Section 14].

ESK packets. The *Encrypted Session Key (ESK) packet*, as the name suggests, contains an encrypted symmetric session key for decrypting the SEIPD packet. There are two types of ESK packets in OpenPGP: the *Public-Key Encrypted Session Key (PKESK) packet* [35, Section 5.1], and the *Symmetric-Key Encrypted Session Key (SKESK) packet* [35, Section 5.3].

The SKESK packet contains a session key, encrypted with or derived from a passphrase. The packet body contains a one-byte version number (4, the only version in [35]), a string-to-key (S2K) specifier, a one-byte symmetric-key algorithm identifier, optionally followed by the encrypted session key; see also Table 4. The S2K specifier determines the key derivation procedure from the passphrase. If the encrypted session key is not present in the SKESK packet, then the key derived from the passphrase is used as the session key, together with the symmetric-key algorithm specified in the SKESK packet. Otherwise, the key is used to decrypt the encrypted session key under the specified key algorithm, using the CFB mode with the all-zero IV. Note that the random block and the "quick check" bytes are not used in the encrypted session key. The decryption recovers the session key, along with the one-byte symmetric algorithm identifier for the session key.

B Forward and Backward Verification

The newer versions of Delta Chat core (1.133) redefine the concept of contact verification.[20] Satisfying the requirements in Section 3.2.1 only means the contact is forward-verified, which allows the contact to send messages to the user in verified chats. The one-on-one chat with the forward-verified contact is also marked as verified. However, in order to get a green checkmark, the contact still needs to get backward-verified, i.e. indicate that the contact has also verified the user's key, which may require the contact to send an additional message to the user. Contacts introduced through the Verified Group

[20] https://github.com/deltachat/deltachat-core-rust/pull/5089

protocol are automatically marked as backward-verified.

This change only affects one of our attacks. Note that verification through gossips in the Verified Group protocol is not affected by the change, where the peer in the gossip would also be marked as backward-verified; therefore, the gossip key injection attack still achieves the same effect for newer Delta Chat versions without any modification.

Impact on the attack in Section 4.4. The attacker would need to send one more message to the target, as if sent from the peer through the verified one-on-one chat with the target, in order to make the target verify the peer. In more detail, the peer chosen by the attacker (which can be the attacker themselves) is forward-verified to the target at this point, but not necessarily backward-verified; that is, the target's client has verified the peer's key, but is not sure if the peer's client has also marked the target's key as verified. In that case, Delta Chat would only mark the one-on-one chat with the peer as verified, but would not mark the peer as verified until it receives a message through the verified chat.

C Setup Contact Protocol Details

We first introduce the notation and simplified primitives for the Setup Contact protocol (Section 3.2.2):

- $\mathsf{pk} = (\mathsf{pk}_{\mathsf{prim}}, \sigma_{\mathsf{prim}}, \mathsf{pk}_{\mathsf{sub}}, \sigma_{\mathsf{sub}})$: an OpenPGP transferable public key, where $\mathsf{pk}_{\mathsf{prim}}$ is the primary public key for signing, σ_{prim} is a self-signature, $\mathsf{pk}_{\mathsf{sub}}$ is the public subkey for encryption, and σ_{sub} is a binding signature on $\mathsf{pk}_{\mathsf{sub}}$ by the primary key. We ignore fields like the User ID that are irrelevant for the protocol.
- $\mathsf{sk} = (\mathsf{pk}, \mathsf{sk}_{\mathsf{prim}}, \mathsf{sk}_{\mathsf{sub}})$: an OpenPGP secret key, where $\mathsf{sk}_{\mathsf{prim}}$ and $\mathsf{sk}_{\mathsf{sub}}$ are private keys for $\mathsf{pk}_{\mathsf{prim}}$ and $\mathsf{pk}_{\mathsf{sub}}$ in pk respectively.
- $\mathsf{fp} = \mathtt{FP}(\mathsf{pk})$: an OpenPGP key fingerprint, which is a 20-byte SHA-1 digest derived from the primary key $\mathsf{pk}_{\mathsf{prim}}$.
- $\mathtt{Seal}(\mathsf{sk}_A, \mathsf{pk}_B, m)$: the Autocrypt "encrypt" process (sign-compress-encrypt), where the message m is signed with sk_A and encrypted with pk_B.
- $\mathtt{Open}(\mathsf{sk}_B, \mathsf{pk}_A, c)$: the Autocrypt "decrypt" process (decrypt-decompress-verify).
- $\mathsf{From} = (\mathsf{name}, \mathsf{addr})$: a `From` header, where name is a name (e.g. `"Alice"`) and addr is an e-mail address (e.g. `"alice@example.org"`).
- peers: the Autocrypt peer state table, which contains each peer's Autocrypt public key, gossip key, verified key, and the timestamps for the keys (see Section 3.1.4). We use $\mathsf{peers}[\mathsf{addr}]$ to refer to the peer state for the contact with address addr, and use $\mathsf{peers}.\mathtt{lookup}(\mathsf{fp})$ to refer to the peer state whose Autocrypt key fingerprint is fp; both can uniquely identify a peer state in Delta Chat.
- INVITE and AUTH: random tokens in the QR code.
- tokens: the token database. We use $\mathsf{tokens}.\mathtt{top}()$ to denote selecting the most recently created INVITE and AUTH tokens from the token database.

Figure 4 shows a simplified run of the Setup Contact protocol. For the purpose of introducing the protocol, it suffices to treat the internals of OpenPGP as a black box. Alice's invite code contains her fingerprint fp_A, address addr_A, nickname name_A, and two random tokens, INVITE and AUTH, taken from the tokens_A database. These tokens do not expire unless Alice manually revokes the corresponding QR code. According to the SecureJoin document, INVITE and AUTH should each be at least 8 bytes. In Delta Chat, both tokens are random strings of 11 Base64 characters.

Remarks. Note that Alice in the protocol is stateless except for maintaining peer states, while Bob is stateful. A user may participate in multiple protocol instances at the same time in the role of Alice, but can only have a single protocol instance at the same time in the role of Bob. Protocol participants find their peer's state in the peer state table using the peer's fingerprint. A fingerprint can uniquely identify a peer in the table, since Delta Chat does not allow two peers to have the same public key fingerprint in the peer state table.

D Options for the Setup Forgery Attack

In this section, we give the detailed options for implementing the attack in Section 4.5. Recall from Appendix A that the body of a version 4 SKESK packet consists of a one-byte version number, a one-byte symmetric algorithm, a string-to-key (S2K) specifier, and, optionally, the encrypted session key. Note that when decrypting an SEIPD packet with the plaintext algorithm, the MDC is still checked.

Option 1: No esk with plaintext data. The attacker can use an SKESK packet with the algorithm set to "plaintext" and without the encrypted session key, followed by some plaintext data with two random bytes prefixed and an MDC packet attached. This way, Delta Chat always decrypts the message to the attacker-chosen data under the plaintext algorithm, and the session key derived from the passphrase is not used at all. As a result, no matter what passphrase the target enters, the malicious Autocrypt Setup Message always successfully decrypts to the malicious key.

Option 2: Modified esk with plaintext data. This option requires the attacker to intercept the target's Autocrypt Setup Message. The attacker takes the SKESK packet from the intercepted Autocrypt Setup Message, and changes the inner algorithm specifier from a known byte (7 for AES-128 in

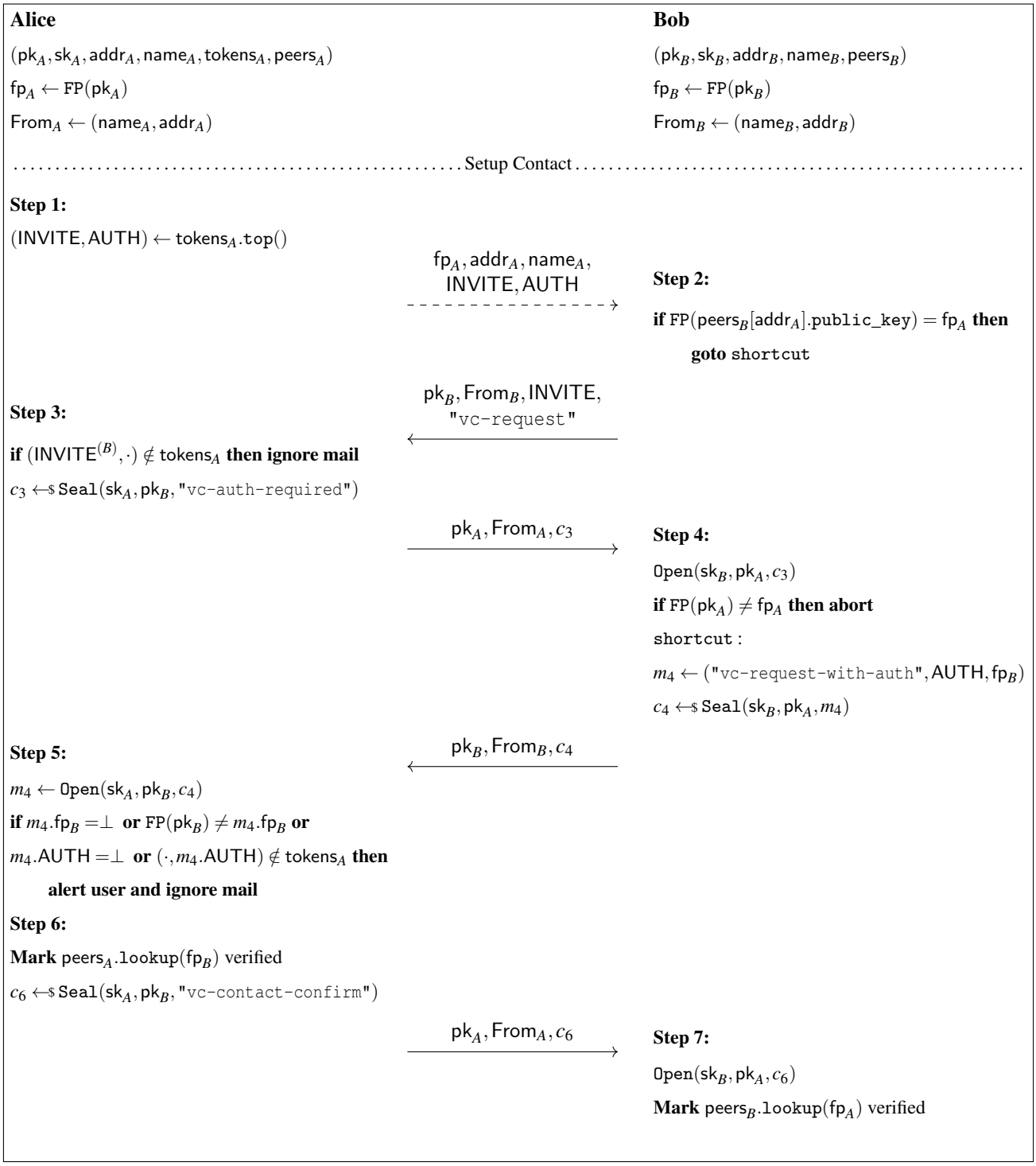

Figure 4: A simplified description of the Setup Contact protocol in Delta Chat. We assume messages on the wire are correctly formatted. Delta Chat checks message labels before possible signature verification and ignores messages with incorrect labels. If `Open` fails on verification, then Alice's client alerts the user and ignores the message, while Bob's client aborts the protocol; for all other `Open` failures, including during decryption and decompression, Delta Chat ignores the message. We do not include fields that are sent by honest participants but are not checked by their peers' clients. We also omit the Autocrypt key update process (Section 3.1), implicitly assuming that the effective dates for outbound messages are increasing for each participant.

Delta Chat) to 0 (plaintext). As there are no integrity checks over the symmetrically-encrypted session key, this can be done by simply xoring the first byte of the encrypted session key with an appropriate byte (7 in our example). Then, the attacker appends the malicious key to the SKESK packet, similar to Option 1. If the target enters the correct Setup Code intended for the intercepted Setup Message, Delta Chat decrypts the encrypted session key algorithm to "plaintext", and the Setup Message decrypts successfully to the malicious key. Otherwise, if the target enters a wrong passphrase, then with probability around 255/256, Delta Chat decrypts the encrypted session key algorithm to a different value, which either does not have a corresponding algorithm, or results in a decryption failure when it is used to decrypt the subsequent SEIPD packet. This option makes the attack harder to detect by users who may suspect manipulation, since the Autocrypt Setup Message no longer decrypts successfully with incorrect passphrases, entered either intentionally or by accident.

Option 3: Plaintext esk with encrypted data. It is also possible for the attacker to set the symmetric algorithm to plaintext in the SKESK packet, such that no matter what passphrase the target enters, the encrypted session key always decrypts to itself. The attacker simply uses the fixed session key to encrypt the malicious key, so that Delta Chat always decrypts the message to the malicious key, irrespective of the passphrase used by the target. This option is harder to detect by inspection of the encrypted e-mail, since only one byte of non-random data is "encrypted" with the plaintext algorithm.

E Attacks Outside of the Threat Model

Here we discuss attacks that are outside of Delta Chat's specified threat model. For example, some attacks may require that the attacker be verified to the target, or break some security properties that are not explicitly claimed to hold in Delta Chat. These attacks may or may not be considered as real threats to Delta Chat users. Note that attacks in the previous subsections may be used to gain privileges for some of these attacks.

Missing standard properties. Note that the threat model outlined in Section 2 excludes attacks that would in general prevent making formal security claims standard in the academic literature on messaging but also more fundamentally on key exchange and secure channels. For example, a more robust protocol would be able to prevent reordering, replay[21] and deletion attacks, but also unknown key-share [16] or identity misbinding attacks such as the one below.

Contact misbinding. In the Setup Contact protocol, while Bob's message carries a protected `From` header, Alice's client still accepts the message and ignores the protected `From` if it is different from the outer unprotected `From`. Therefore, a network attacker could perform an identity-misbinding attack between Alice and Bob, where the attacker modifies messages from Bob to Alice as if they were sent from another address controlled by the attacker, and forwards messages intended for the attacker-controlled address to Bob. Upon completion of the Setup Contact protocol, Alice's client would mark the attacker-controlled address as verified with a green checkmark, but still under Bob's public key.

To fix this issue, Delta Chat should ensure that the protected `From` header is present in SecureJoin messages whenever possible, and reject the message with possible warnings to the user if it is missing or different from the unprotected `From`. Note that simply preferring the protected `From` would not be a correct mitigation as it would make spoofing easier for an attacker.

Attacks on privacy. An eavesdropping attacker can easily distinguish Autocrypt traffic by checking the Autocrypt header. The attacker can also distinguish messages from different groups, since the group ID is a part of the plaintext `Message-ID` header. An attacker that can only observe and modify partial network traffic, e.g. a malicious e-mail server, may "taint" Autocrypt keys in order to to learn more about the social graph of the target. The attacker can do this by adding unhashed subpackets to OpenPGP keys in Autocrypt headers found in network messages, which is possible since these fields are not protected by signatures nor contribute to the key fingerprint. Afterwards, the attacker can interact with potential contacts and see from the `Autocrypt-Gossip` header whether they possess the tainted key.

Insider Attacks

Another class of attacks excluded from Delta Chat's threat model is the case of insider attacks, where the attacker is able to corrupt one of the contacts of the target, or is part of the same verified group as the target. While preventing all such attacks may seem impossible, there are scenarios in which protocols can be more or less vulnerable, and which may also have real consequences. Below, we list an example applicable to Delta Chat's Setup Contact protocol. Note that an attacker who is instead part of a verified group with the target could achieve the same results, as the Verified Group protocol is trivially vulnerable to insider attacks.

Another header manipulation attack. Suppose Mallory first performs the Setup Contact protocol with Bob or captures a `vc-request-with-auth` message from Bob. Then, Mallory obtains Alice's QR code. Mallory could verify Bob to Alice by forwarding the `vc-request-with-auth` message she received or captured from Bob to Alice and adding a `Secure-Join-Auth` plaintext header with Alice's AUTH token to the message, which would overwrite the protected `Secure-Join-Auth` header. Since Bob's message does not

[21] Newer versions of Delta Chat do add limited replay protection by including a `Message-ID` header in the protected payload, see Section 4.2.

properly bind to Mallory, on receiving the message, Alice's client would genuinely believe Bob is establishing contact with her and mark Bob's key as verified.

Setting `Secure-Join-Auth` as a secured header would solve this issue.

F Miscellaneous Issues

We list weaknesses or issues that are not vulnerabilities per se, but may nonetheless be of interest.

Compression side channels. In Delta Chat, compression is enabled by default for OpenPGP messages, even for automated e-mails like SecureJoin and synchronisation messages. However, OpenPGP performs compression before encryption, which is known to be dangerous since compression leaks information about the plaintext [42], as evidenced by the infamous CRIME and BREACH attacks [36, 54]. Delta Chat is potentially vulnerable to compression side-channel attacks, especially since some messages may co-locate secrets and attacker-controlled fields, such as synchronisation messages. However, there appears to not be an efficient way for the attacker to utilise the compression side channels in Delta Chat. Delta Chat should disable compression for SecureJoin and synchronisation messages, and consider disabling compression for all messages.

Weak keys and algorithms. While Delta Chat defaults to a strong set of OpenPGP configurations, the rPGP library accepts old packet formats and can handle some outdated ciphers, like IDEA and CAST5.

Logging. As noted in [57], Delta Chat may write sensitive information to unencrypted log files in local storage. We found that Delta Chat continues to log sensitive information, such as the QR code used in the SecureJoin protocols and error information in the cryptographic libraries.

Format oracles. Some OpenPGP implementations return detailed error information that could be used to instantiate a format oracle [41, 46]. This is also the case for rPGP. For example, the rPGP library still checks the "quick check" bytes, and returns a distinct error if the quick check fails. The security audit on rPGP marked this issue as a medium-risk finding [56], which had previously been left open and was eventually fixed as a result of our analysis.[22] Since Delta Chat logs errors in rPGP verbatim most of the time, an attacker with access to the target's log file may be able to recover encrypted message contents from the target, using the error information in the log-file as "lunchtime" format oracles. However, since Delta Chat stores messages and private keys unencrypted, having access to the target's machine would already break most security guarantees of Delta Chat.

Partial signing oracle. In the Setup Contact protocol, Alice shows a QR code to Bob, which contains a randomly generated AUTH token. The SecureJoin document does not specify the format or the maximum length of the AUTH token; Delta Chat generates AUTH tokens of 11 Base64 characters, but it also accepts tokens that are very long or contain some non-Base64 characters, such as space, comma and period. Therefore, a malicious Alice could make the AUTH token in the QR code resemble an authentic English text, which is difficult for Bob to notice. After scanning Alice's QR code, Bob's client at some point sends a `vc-request-with-auth` message to Alice, with the AUTH token in the signed and encrypted payload. After decryption and decompression, Alice obtains Bob's signature on a message that she has partial control of, which Alice could potentially use to incriminate Bob. However, other fields inside the signed message would make it apparent that it was a protocol message.

Key expiration time. The key expiration time in OpenPGP keys is currently not used in Delta Chat. All keys created in Delta Chat have no expiration time, and nothing prevents the usage of expired Autocrypt keys in Delta Chat. This is a deliberate decision of the Delta Chat team as they believe encrypting a message under an expired key is better than not encrypting it. Note that key expiration only protects against the leakage of subkeys and not the primary key.

Domain separation. Recall that the Verified Group protocol (Section 3.2.3) associates different random tokens with different groups. However, Delta Chat seems to treat tokens as valid in the Verified Group protocol even if they are associated with different groups or intended for the Setup Contact protocol.

TOCTOU attack. There seems to be a possibility of time-of-check to time-of use (TOCTOU) attacks on Delta Chat. More specifically, in the Setup Contact protocol, on receiving the `vc-auth-required` message from Alice (step 4), Bob first checks that the fingerprint of Alice's Autocrypt key matches the fingerprint in the QR code, and then encrypts the AUTH token under Alice's Autocrypt key. If Mallory managed to spoof a message from Alice to Bob with a malicious Autocrypt header, then it is theoretically possible that Bob's client would update Alice's Autocrypt key to the malicious one after the check but before encryption, and therefore Bob's client could accidentally encrypt the AUTH token under a key controlled by Mallory. However, this attack is likely impractical since the time interval between check and use is negligible and e-mails have a low granularity in time.

Unknown verification attack. It appears that Bob's messages in the Setup Contact protocol do not properly bind to Alice's public key or fingerprint. Even though the `vc-request-with-auth` message from Bob carries Alice's public key and address as an `Autocrypt-Gossip` header in the protected payload, Alice's client does not reject the message if the gossiped key is different from Alice's own key. If a network attacker Mallory obtained Alice's QR invite code, she could modify her own QR code to contain the same tokens as in Alice's QR code while keeping other fields unchanged, and convince Bob to scan the modified QR code, possibly as an insider attacker. Mallory then forwards messages from

[22] https://github.com/rpgp/rpgp/issues/183

Bob to Alice, re-encrypted under Alice's public key, and crafts appropriate responses for Bob on her own. At the end of the Setup Contact protocol, Bob's client would mark Mallory as verified, but Alice's client would mark Bob as verified, even though Bob did not intend to establish contact with Alice.

In Delta Chat, Alice's client should check that SecureJoin messages from Bob carry Alice's address and public key in the protected payload as a gossip, except for the very first `vc/vg-request` message, and reject the message with possible warnings if not.