Autocrypt aims to incrementally and carefully replace cleartext e-mail with end-to-end encrypted e-mail. This differs from the traditional approach of maximizing the security of individual mail communications. Sometimes Autocrypt recommends to send cleartext mail even though encryption appears technically possible. This is because we want to avoid unreadable mail for users. Users may mix both Autocrypt-capable and traditional mail apps and they may lose devices or in other ways the ability to decrypt in unrecoverable ways. Reverting to cleartext when we suspect such situations is a key part of our aim to stay out of the way of users.

Another major difference in approach is that Autocrypt Level 1 only defends against passive data collection attacks. We share and support the new perspective stated in RFC7435 (“Opportunistic Security: Some Protection Most of the Time”). Protection against active adversaries (those which modify messages in transit) is the aim of future specifications.

Level 1 makes it easy for users to encrypt, based on an automatic and decentralized key distribution mechanism. There are no dependencies on key servers and it is meant to work with existing e-mail providers. Level 1 focuses on the use of Autocrypt on a single device. Users get rudimentary support on using Autocrypt on more than one device or mail app. This is internally realized through sending and receiving an Autocrypt Setup Message, secured by manually entering a long number. Improving usability for maintaining synchronized Autocrypt state on multiple devices is the aim of future specification efforts.

Last but not least, Level 1 is meant to be relatively easy for developers to adopt. It describes the basic capabilities required for a mail app to be Autocrypt-capable at Level 1, allowing it to exchange end-to-end encrypted e-mails with other Autocrypt-capable mail apps. The spec contains detailed guidance on protocol, internal state and user interface concerns. We have a good track record of supporting new implementers. Please don’t hesitate to contact the group or bring up issues or pull requests. Autocrypt is a living specification and we envision both bugfix and backward-compatible feature releases.

1 https://tools.ietf.org/html/rfc7435.html#section-1.2
2 https://autocrypt.org/en/latest/contact.html
1 Terminology

1.1 Keywords to indicate requirement levels

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in the IETF’s Best Current Practice 14 (as defined in RFC 2119 and RFC 8174) when, and only when, they appear in all capitals, as shown here.

2 Overview

2.1 Approach and High Level Overview

Autocrypt’s primary goal is to automate both secret and public key management so that users can encrypt mail without specialized knowledge.

This specification adds an Autocrypt-specific mail header (page 5) to outgoing mails, which contains, among other information, the sender’s public key. Transferring public keys in-band means that key discovery in Autocrypt does not require external infrastructure like OpenPGP keyservers or x509 PKI.

Autocrypt provides a set of rules (page 7) that tracks this information for each communication peer. Autocrypt uses this information to determine whether encryption is possible and makes a recommendation (page 8) about whether encryption should be enabled for a given set of recipients.

This specification also introduces the Autocrypt Setup Message (page 13) as a way to transfer secret key material and related settings to other e-mail programs controlled by the same user. This spec also provides guidance on how and when to generate (page 14), look for (page 16), and import (page 15) these messages.

Autocrypt aggressively distributes public keys, but conservatively recommends encryption to avoid disruption to established e-mail workflows. Specifically, Autocrypt only recommends that an e-mail be encrypted if encryption is possible, and:

1) The sender specifically requests encryption during message composition;
2) The e-mail is in reply to an encrypted message; or,
3) The sender and the recipients have explicitly stated that they prefer (page 5) encrypted e-mail.

2.2 Requirements on MUA/E-mail Provider interactions

Autocrypt tries to impose minimal requirements on MUA and e-mail service interactions. Specifically, an Autocrypt-capable MUA needs to be able to:

• Control the contents of outgoing e-mail including the ability to set custom e-mail headers;
• Send e-mail on its own (required by the Autocrypt Setup Message (page 13));
• Read whole, raw e-mails including message headers; and,
• Optionally, scan the user’s mailbox for mail with specific headers.

If a particular e-mail account does not expose one of the required features (e.g., if it only exposes a javascript-driven web interface for message composition that does not allow setting e-mail headers), then the e-mail account cannot be used with Autocrypt. An Autocrypt-capable MUA may still access and control the account, but it will not be able to enable Autocrypt on it.

---


Autocrypt Specification 1.1.0
2.3 Autocrypt Internal State

An Autocrypt MUA needs to associate information with the peers it communicates with and the accounts it controls.

2.3.1 Communication Peers

Each communication peer is identified by an e-mail address. Autocrypt associates state with each peer. Conceptually, we represent this state as a table named `peers`, which is indexed by the peer’s canonicalized e-mail address (page 17).

For the peer with the address `addr`, an MUA MUST associate the following attributes with `peers[addr]`:

- `last_seen`: The UTC timestamp of the most recent effective date (definition (page 7)) of all messages that the MUA has processed from this peer.
- `autocrypt_timestamp`: The UTC timestamp of the most recent effective date (the “youngest”) of all messages containing a valid Autocrypt header that the MUA has processed from this peer.
- `public_key`: The value of the `keydata` attribute derived from the youngest Autocrypt header that has ever been seen from the peer.
- `prefer_encrypt`: The `prefer-encrypt` value (either `nopreference` or `mutual`) derived from the youngest Autocrypt header ever seen from the peer.

Autocrypt-capable MUAs that implement Gossip (page 10) should also associate the following additional attributes with `peers[addr]`:

- `gossip_timestamp`: The UTC timestamp of the most recent effective date of all messages containing a valid Autocrypt-Gossip header about the peer.
- `gossip_key`: The value of the `keydata` attribute derived from the most recent message containing a valid Autocrypt-Gossip header about the peer.

How this information is managed and used is discussed in Peer State Management (page 5).

2.3.2 Accounts controlled by the MUA

A Level 1 MUA maintains an internal structure `accounts` indexed by the account’s canonicalized e-mail address (page 17) (`addr`). For each account controlled by the MUA, `accounts[addr]` has the following attributes:

- `enabled`: a boolean value, indicating whether Autocrypt is enabled for this account.
- `secret_key`: The secret key material used for the account (see Secret key generation and storage (page 12)).
- `public_key`: The OpenPGP transferable public key (OpenPGP “Transferable Public Key”5) derived from the secret key.
- `prefer_encrypt`: The user’s encryption preference for this account. This is either `mutual` or `nopreference`. This SHOULD default to `nopreference`.

If `accounts[addr].enabled` is `true`, the MUA SHOULD allow the user to switch the setting for `accounts[addr].prefer_encrypt`. This choice might be hidden in something like a “preferences pane”. See Account Preferences (page 16) for a specific example of how this could look.

How this information is managed and used is discussed in Managing accounts controlled by the MUA (page 12).

---

5 https://tools.ietf.org/html/rfc4880.html#section-11.1
3 Peer State Management

An Autocrypt MUA updates the state it holds for each communication peer using the e-mails received from that peer. Specifically, Autocrypt updates the state using the Autocrypt e-mail header.

3.1 The Autocrypt Header

The Autocrypt header has the following format:

Autocrypt: addr=a\@b.example.org; [prefer-encrypt=mutual;] keydata=BASE64

There are three defined attributes:

- The `addr` attribute is mandatory, and contains the single recipient address this header is valid for. If this address differs from the one in the `From` header, the entire Autocrypt header MUST be treated as invalid.

  The Internet Message Format documents three types of originator fields: `From`, `Sender`, and `Reply-To`. Autocrypt is concerned only with the `From` field, and ignores the other originator fields.

- The `prefer-encrypt` attribute is optional and can only occur with the value `mutual`. Its presence in the Autocrypt header indicates an agreement to enable encryption by default with other peers who have the same preference. An Autocrypt Level 1 MUA that sees the attribute with any other value (or that does not see the attribute at all) should interpret the value as `nopreference`.

- The `keydata` attribute is mandatory, and contains the key data for the specified `addr` recipient address. The value of the `keydata` attribute is a Base64 representation of the binary OpenPGP “Transferable Public Key”. For ease of parsing, the `keydata` attribute MUST be the last attribute in this header.

Additional attributes are possible before the `keydata` attribute. If an attribute name starts with an underscore (_), it is a “non-critical” attribute. An attribute name without a leading underscore is a “critical” attribute. The MUA SHOULD ignore any unsupported non-critical attributes and continue parsing the rest of the header as though the attribute does not exist. It MUST treat the entire Autocrypt header as invalid if it encounters a “critical” attribute that it doesn’t support.

To introduce incompatible changes, future versions of Autocrypt may send multiple Autocrypt headers, and hide the incompatible headers from Level 1 MUAs by using critical attributes. According to the above rules, such headers will be judged invalid, and discarded by level 1 MUAs. Such an update to the specification will also have to describe how an updated MUA will deal with multiple valid headers.

A MUA MUST NOT send an Autocrypt header above 10 KiB in size (including the Autocrypt: prefix and folding whitespace). Larger headers are likely the result of unintentionally bloated OpenPGP data. To ensure a consistent failure mode across clients, MUAs SHOULD treat such headers as invalid.

Note:

- An Autocrypt header following this specification will be up to 3 KiB in size. Some users might import 4096-bit RSA keys, which may result in an Autocrypt header of up to 5 KiB.

- The 10 KiB limit was chosen to accommodate for these use cases but also avoid headers being truncated or emails being rejected by MTAs with a 10 KiB header size limit.

3.1.1 OpenPGP Based key data

The `keydata` sent by an Autocrypt-enabled Level 1 MUA MUST consist of an OpenPGP “Transferable Public Key” containing exactly these five OpenPGP packets:

- a signing-capable primary key

---


Autocrypt Specification 1.1.0
• a user id
• a self signature over the user id by the primary key
• an encryption-capable subkey
• a binding signature over the subkey by the primary key

The content of the user id packet is only decorative. By convention, it contains the same address used in the `addr` attribute placed in angle brackets. (This makes it conform to the RFC 5322⁹ grammar `angle-addr`.) For compatibility concerns, the user id SHOULD NOT be an empty string.

These packets MUST be assembled in binary format (not ASCII-armored), and then base64-encoded.

A Level 1 MUA MUST be capable of processing and handling Ed25519 public keys for signatures, as well as Cvs25519 for encryption. It MAY support other OpenPGP key formats found in an Autocrypt header (for example, by passing it agnostically to an OpenPGP backend for handling). In particular, it SHOULD support the RSA algorithm for both signatures and encryption.

Note: To optimize for compatibility, earlier versions of this document REQUIRED support for RSA, and recommended a 3072 bit RSA key configuration. Support for elliptic curve cryptography in deployed OpenPGP implementations improved since then, and the switch to ECC was made in version 1.1 (January 2019) due to message size considerations.

3.1.2 Header injection in outbound mail

During message composition, if the `From:` header of the outgoing e-mail (the `from-addr`) matches an address for which `accounts[from-addr].enabled` is true and the Autocrypt-capable MUA has secret key material (`accounts[from-addr].secret_key`), the MUA SHOULD include an Autocrypt header.

This header MUST contain the corresponding public key material (`accounts[from-addr].public_key`) as the `keydata` attribute, and `from-addr` as the `addr` attribute. The most minimal Level 1 compliant MUA will only include these two attributes. If `accounts[from-addr].prefer_encrypt` is set to `mutual`, then the header MUST have a `prefer-encrypt` attribute with the value `mutual`.

The MUA MUST NOT include more than one valid Level 1 Autocrypt header (see Updating Autocrypt Peer State (page 7)).

If the `From` address changes during message composition (e.g., if the user selects a different outbound identity), then the MUA MUST change the Autocrypt header accordingly.

An MUA SHOULD send out the same Autocrypt: header in all messages from a given outbound identity. An MUA SHOULD NOT vary the header based on the message’s recipients. If (for whatever reason) the MUA needs to update (or discovers an update of) the user’s `keydata` at some point, the MUA SHOULD send the updated `keydata` in all subsequent Autocrypt headers.

See Example Autocrypt headers (page 18) for examples of outbound headers and the following sections for header format definitions and parsing.

3.2 Internal state storage

See Communication Peers (page 4) for the information stored for each communication peer.

Autocrypt MUAs keep state about each peer, to handle several nuanced situations that have caused trouble or annoyance in the past. This state is updated even when the peer sends mail without an Autocrypt header.

For example, if a remote peer disables Autocrypt or drops back to only using a non-Autocrypt MUA, we must stop sending encrypted mails to this peer automatically.


Autocrypt Specification 1.1.0
In addition to the per-peer state described in Communication Peers (page 4), MUAs MAY also store other information gathered for heuristic purposes, or for other cryptographic schemes (see the Autocrypt website\(^{10}\) for some example ideas).

However, in order to support future synchronization of Autocrypt state between MUAs, it is critical that Autocrypt-capable MUAs maintain the state specified here, regardless of what additional state they track.

**Note:**

- An implementation MAY also choose to use keys from other sources (e.g., a local keyring) at its own discretion.
- If an implementation chooses to automatically ingest a key from an application/pgp-keys attachment as though it was found in an Autocrypt header, it should only do so if the attached key has a User ID\(^{11}\) that matches the message’s From address.

### 3.3 Updating Autocrypt Peer State

Incoming messages may be processed to update the peers entry for the sender identified by from-addr as extracted from the From header, by an MUA at receive or display time.

Messages SHOULD be ignored (i.e., peers[from-addr] SHOULD NOT be updated) in the following cases:

- The content-type is multipart/report. In this case, it can be assumed the message was auto-generated. This avoids triggering a reset state from received Message Disposition Notifications (RFC 3798\(^{12}\)).
- There is more than one address in the From header.
- The MUA believes the message to be spam. If the user marks the message as not being spam the message MAY then be processed for Autocrypt headers.

When parsing an incoming message, an MUA SHOULD examine all Autocrypt headers, rather than just the first one. If there is more than one valid header, this SHOULD be treated as an error, and all Autocrypt headers discarded as invalid.

Updating peers[from-addr] depends on:

- the effective date of the message, which we define as the sending time of the message as indicated by its Date header, or the time of receipt if that date is in the future or unavailable.

**Note:** A message without a Date header, or with a Date that seems to be in the far future can cause problems for MUAs that encounter the message repeatedly (e.g., re-delivery, subsequent scans, etc). An MUA MAY decide to ignore such a message entirely for the purposes of Autocrypt processing. If an MUA is capable of associating information with a received message, it could instead save the effective date of such a message the first time it sees it to avoid accidental re-processing.

- the keydata and prefer-encrypt attributes of the single valid Autocrypt header (see above), if available.

The update process proceeds as follows:

1. If the message’s effective date is older than the peers[from-addr].autocrypt_timestamp value, then no changes are required, and the update process terminates.
2. If the message’s effective date is more recent than peers[from-addr].last_seen then set peers[from-addr].last_seen to the message’s effective date.
3. If the Autocrypt header is unavailable, no further changes are required and the update process terminates.

\(^{10}\) https://autocrypt.org/en/latest/optional-state.html

\(^{11}\) https://tools.ietf.org/html/rfc4880.html#section-5.11

4. Set `peers[from-addr].autocrypt_timestamp` to the message’s effective date.

5. Set `peers[from-addr].public_key` to the corresponding `keydata` value of the Autocrypt header.

6. Set `peers[from-addr].prefer_encrypt` to the corresponding `prefer-encrypt` value of the Autocrypt header.

### 3.4 Provide a recommendation for message encryption

On message composition, an Autocrypt-capable MUA can decide whether to try to encrypt the new e-mail message. Autocrypt provides a recommendation for the MUA.

All Autocrypt-capable MUAs should be able to calculate the same Autocrypt recommendation. This recommendation algorithm provides sensible guidance that avoids many common problems, and Autocrypt-capable MUAs SHOULD follow the recommendation. An implementation that deviates from the recommendation should do so on the basis of specific external evidence or knowledge, while carefully considering the impact of any variation, including:

- does it increase the chance of producing unexpectedly unreadable mail (for either the sender or the recipient)?
- does it leak previously encrypted content in the clear?
- does it force the user to confront a choice they do not have the information or knowledge to make safely?

If an implementation deviates from the Autocrypt recommendation in a meaningful and useful way, the implementer should describe the variation publicly so it can be considered for future revisions of this specification.

#### 3.4.1 Recommendation structure

The Autocrypt recommendation depends on the recipient addresses of the draft message, and on whether or not the message is a reply to an encrypted message. When the user changes the recipients during composition, the Autocrypt recommendation may change.

The output of the Autocrypt recommendation algorithm has two elements:

- **ui-recommendation**: a single state recommending the state of the encryption user interface, described below.
- **target-keys**: a map of recipient addresses to public keys.

`ui-recommendation` can take four possible values:

- **disable**: Disable or hide any UI that would allow the user to choose to encrypt the message. This happens iff encryption is not immediately possible.
- **discourage**: Enable UI that would allow the user to choose to encrypt the message, but do not default to encryption. If the user manually enables encryption, the MUA SHOULD warn that the recipient may not be able to read the message. This warning message MAY be supplemented using optional counters and user-agent state\(^\text{13}\).
- **available**: Enable UI that would allow the user to choose to encrypt the message, but do not default to encryption.
- **encrypt**: Enable UI that would allow the user to choose to send the message in cleartext, and default to encryption.

#### 3.4.2 Recommendations for single-recipient messages

The Autocrypt recommendation for a message composed to a single recipient with the e-mail address `to-addr` depends primarily on the value stored in `peers[to-addr]` (page 4).

\(^\text{13}\) [Autocrypt.org](https://autocrypt.org/en/latest/optional-state.html)
Determine if encryption is possible

If there is no peers[to-addr], then set ui-recommendation to disable, and terminate.

For the purposes of the rest of this recommendation, if either public_key or gossip_key is revoked, expired, or otherwise known to be unusable for encryption, then treat that key as though it were null (not present).

If both public_key and gossip_key are null, then set ui-recommendation to disable and terminate.

Otherwise, we derive the recommendation using a two-phase algorithm. The first phase computes the preliminary-recommendation.

Preliminary Recommendation

If public_key is null, then set target-keys[to-addr] to gossip_key and set preliminary-recommendation to discourage and skip to the Deciding to Encrypt by Default (page 9).

Otherwise, set target-keys[to-addr] to public_key.

If autocrypt_timestamp is more than 35 days older than last_seen, set preliminary-recommendation to discourage.

Otherwise, set preliminary-recommendation to available.

Deciding to Encrypt by Default

The final phase turns on encryption by setting ui-recommendation to encrypt in two scenarios:

- If preliminary-recommendation is either available or discourage, and the message is composed as a reply to an encrypted message, or
- If the preliminary-recommendation is available and both peers[to-addr].prefer_encrypt and accounts[from-addr].prefer_encrypt are mutual.

Otherwise, the ui-recommendation is set to preliminary-recommendation.

3.4.3 Recommendations for messages to multiple addresses

For level 1 MUAs, the Autocrypt recommendation for a message composed to multiple recipients, we derive the message’s recommendation from the recommendations for each recipient individually.

The aggregate target-keys for the message is the merge of all recipient target-keys.

The aggregate ui-recommendation for the message is derived in the following way (the earliest matching rule encountered below takes precedence over later rules):

1. If any recipient has a ui-recommendation of disable, then the message’s ui-recommendation is disable.
2. If every recipient has a ui-recommendation of encrypt, then the message ui-recommendation is encrypt.
3. If any recipient has a ui-recommendation of discourage, then the message ui-recommendation is discourage.

Otherwise, the message ui-recommendation is available.

While composing a message, a situation might occur where the ui-recommendation is available, the user has explicitly enabled encryption, and then modifies the list of recipients in a way that changes the ui-recommendation to disable. When this happens, the MUA should not disable encryption without
communicating this to the user. A graceful way to handle this situation is to save the enabled state, and only prompt the user about the issue when they send the mail.

### 3.5 Message Encryption

**Note:** An e-mail that is said to be “encrypted” here will be both signed and encrypted in the cryptographic sense.

An outgoing e-mail message will be sent encrypted in either of two cases:

- the Autocrypt recommendation for the list of recipients is encrypt, and not explicitly overridden by the user, or
- the Autocrypt recommendation is available or discourage, and the user chose to encrypt.

When encrypting, the MUA MUST construct the encrypted message as a PGP/MIME message that is signed by the user’s Autocrypt key, and encrypted to the currently known Autocrypt key of each recipient, as well as the sender’s Autocrypt key.

#### 3.5.1 Cleartext replies to encrypted messages

In the common case, a reply to an encrypted message will also be encrypted. Due to Autocrypt’s opportunistic approach to key discovery, however, it is possible that keys for some of the recipients may not be available, and, as such, a reply can only be sent in the clear.

To avoid leaking cleartext from the original encrypted message in this case, the MUA MAY prepare the cleartext reply without including any of the typically quoted and attributed text from the previous message. Additionally, the MUA MAY include some text in the message body describing why the usual quoted text is missing. An example of such copy can be found in Example Copy when a Reply can’t be Encrypted (page 20).

The above recommendations are only “MAY” and not “SHOULD” or “MUST” because we want to accommodate a user-friendly Level 1 MUA that stays silent and does not impede the user’s ability to reply. Opportunistic encryption means we can’t guarantee encryption in every case.

### 3.6 Key Gossip

It is a common use case to send an encrypted mail to a group of recipients. To ensure that these recipients can encrypt messages when replying to that same group, the keys of all recipients can be included in the encrypted payload. This does not include BCC recipients, which by definition must not be revealed to other recipients.

The Autocrypt-Gossip header has the same format as the Autocrypt header (see autocryptheaderformat (page 6)). Its addr attribute indicates the recipient address this header is valid for as usual, but may relate to any recipient in the To or Cc header. See example in Example Autocrypt Gossip headers (page 18).

The Autocrypt-Gossip header MAY also be used to include keys for the address specified in the Reply-To header. This allows replying encrypted even if the address differs from those in the From, To, and Cc headers.

#### 3.6.1 Key Gossip Injection in Outbound Messages

An Autocrypt MUA MAY include Autocrypt-Gossip headers in messages. These headers MUST be placed in the root MIME part of the encrypted message payload. The encrypted payload in this case contains one Autocrypt-Gossip header for each address. Each header:

- MUST include an addr attribute that matches one of the addresses in the To, Cc, or Reply-To headers.

---

• MUST include the keydata attribute. For To and Cc headers it MUST contain the same public key which is used to encrypt the mail to the recipient referenced by addr. See also Preliminary Recommendation (page 9) for how this key is selected. For the address in the Reply-To headers it SHOULD contain the public key which the sender expects to be used for that address.

• SHOULD NOT include a prefer-encrypt attribute.

If a key has multiple user ids, only one SHOULD be contained in keydata. This user id SHOULD be picked to match the addr attribute, if possible. This is only relevant for keys which came from or were merged with data from external sources.

To avoid leaking metadata about a third party in the clear, an Autocrypt-Gossip header SHOULD NOT be added outside an encrypted MIME part.

3.6.2 Updating Autocrypt Peer State from Key Gossip

An incoming message may contain one or more Autocrypt-Gossip headers in the encrypted payload. Each of these headers may update the Autocrypt peer state of the gossiped address identified by its addr value (referred to here as gossip-addr) in the following way:

1. If gossip-addr does not match any address in the mail’s To, Cc, or Reply-To header, the update process terminates (i.e., header is ignored).
2. If peers[gossip-addr].gossip_timestamp is more recent than the message’s effective date, then the update process terminates.
3. Set peers[gossip-addr].gossip_timestamp to the message’s effective date.
4. Set peers[gossip-addr].gossip_key to the value of the keydata attribute.

4 Message Drafts

During message composition, the user may want to close the dialog and resume composition at a later point. To preserve the session state, messages are typically saved as “Drafts” in a special folder. Messages from this folder may also be resumed from a different MUA.

To prevent the cleartext of a message draft from leaking to the provider, the MUA can encrypt the draft message to itself (but not the recipients) before uploading. It MUST encrypt in this way if the message is going to be sent encrypted, and SHOULD encrypt if the user might decide to encrypt at a later point, or otherwise keep the draft local only. Drafts encrypted in this way SHOULD be stored in PGP/MIME format.

A MUA SHOULD ignore the signature status of encrypted drafts, and conversely encrypted drafts do not need to be signed. This allows storage of drafts independently from the user’s secret key.

4.1 Storing Draft State

To store information about whether a message should be encrypted when sent, an Autocrypt-Draft-State header MAY be added to the MIME header of the draft message when it is stored.

The Autocrypt-Draft-State header consists of a list of attributes with the same syntax as the Autocrypt header (page 5) itself, which also supports critical and non-critical attributes following the same semantics. There are three defined attributes, all of which have a binary value of yes or no:

• The critical encrypt attribute specifies whether the message would be sent encrypted or not, at the time of saving the draft. It MUST be present in the header.
• The non-critical _is-reply-to-encrypted attribute indicates whether the message is composed as a reply to an encrypted message. This affects the Autocrypt recommendation (page 9). It is optional and defaults to no.
• The non-critical _by-choice attribute indicates whether the value of the encrypt attribute was made by immediate user choice. It is optional and defaults to no.

Example:

```
Autocrypt-Draft-State: encrypt=yes; _by-choice=yes;
```

The semantics of this header are defined only in the context of a message that is loaded as a draft. It SHOULD NOT be processed for messages loaded in any other context. It MUST be stripped from a message before it is sent.

**Note:** To minimize the amount of exposed metadata, it would be desirable to store this header in the encrypted payload of the message, rather than its outer envelope. However, this causes technical difficulties in some MUAs, which is why it is left out here to optimize for interoperability.

### 4.2 Autocrypt Gossip headers for drafts

When opening a draft that indicates it should be encrypted when sent via the Autocrypt-Draft-State header, the resuming MUA might be lacking keys for recipients that were originally available when the draft was stored.

To solve this problem, a MUA MAY make use of the Key Gossip (page 10) mechanism to include the keys of intended recipients in the draft message. To do so, it simply places one Autocrypt-Gossip header per recipient in the MIME header of the encrypted payload of the draft.

Conversely, when opening a draft, a MUA MAY import keys from Autocrypt-Gossip headers that are present in the MIME headers of the encrypted payload. These headers should update the gossip_key and gossip_timestamp fields of the relevant peers as usual.

### 5 Managing accounts controlled by the MUA

See Accounts controlled by the MUA (page 4) for a definition of the structure of information stored about the MUA’s own e-mail accounts.

#### 5.1 Secret key generation and storage

The MUA SHOULD generate and store a Ed25519 plus Cv25519 secret key for the user, the former for signing and self-certification, the latter for decrypting. The MUA MUST be capable of assembling these keys into an OpenPGP certificate (RFC 4880 “Transferable Public Key”\(^\text{15}\)) that indicates these capabilities.

##### 5.1.1 Secret key protection at rest

The secret key material should be protected from access by other applications or co-tenants of the device at least as well as the passwords the MUA retains for the user’s IMAP or SMTP accounts.

The MUA MAY protect the secret key (and other sensitive data it has access to) with a password, but it SHOULD NOT require the user to enter the password each time they send or receive a mail. Since Autocrypt-enabled MUAs sign all encrypted outgoing messages (page 10), it could happen that the user has to enter the password very often, both for reading and sending mail. This introduces too much friction to become part of a routine daily workflow.

Note that password protection of the secret key carries with it a risk that the user might forget their password, which might result in catastrophic data loss. Unlike IMAP or SMTP credentials (which can be reset by the server operator given some sort of out-of-band confirmation), there is no recovery workflow possible for the loss of a

\(^\text{15}\) https://tools.ietf.org/html/rfc4880.html#section-11.1
password protecting a secret key. An MUA that chooses to offer password protection of the secret key (or other sensitive data) SHOULD support usable and secure backup/recovery workflows for the protected material.

Protection of the user’s keys (and other sensitive data) at rest is achieved more easily and securely with filesystem-based encryption and other forms of access control.

5.2 Handling Multiple Accounts and Aliases

An MUA that is capable of connecting to multiple e-mail accounts SHOULD have a separate and distinct Autocrypt accounts[from-addr] for each e-mail account with the address from-addr.

A multi-account MUA MAY maintain a single peers table that merges information from e-mail received across all accounts for the sake of implementation simplicity. While this results in some linkability between accounts (the effect of mails sent to one account can be observed by activity on the other account), it provides a more uniform and predictable user experience. Any linkability concerns introduced by Autocrypt can be mitigated by using a different MUA for each e-mail account.

Sometimes a user may be able to send and receive e-mails with multiple distinct e-mail addresses (“aliases”) via a single account. For the purposes of Autocrypt, the MUA SHOULD treat each specific alias as a distinct account.

5.3 Avoiding MUA Conflicts

If more than one Autocrypt-enabled MUA generates a key and then distributes it to communication peers, encrypted mail sent to the user is only readable by the MUA that sent the last message. This can lead to behavior that is unpredictable and confusing for the user.

See section Helping Users get Started (page 16) for guidance on how to detect and avoid such a situation.

5.4 Autocrypt Setup Message

To avoid “lock-in” of secret key material on a particular MUA, Autocrypt level 1 includes a way to “export” the user’s keys and her prefer-encrypt state (page 4) for other MUAs to pick up, asynchronously and with explicitly required user interaction.

The mechanism available is a specially-formatted e-mail message called the Autocrypt Setup Message. An already-configured Autocrypt MUA can generate an Autocrypt Setup Message, and send it to itself. A not-yet-configured Autocrypt MUA (a new MUA in a multi-device case, or recovering from device failure or loss) can import the Autocrypt Setup Message and recover the ability to read existing messages.

An Autocrypt Setup Message is protected with a Setup Code (page 14).

5.4.1 Message Structure

The Autocrypt Setup Message itself is an e-mail message with a specific format. While the message structure is complex, it is designed to be easy to pack and unpack using common OpenPGP tools, both programmatically and manually.

- Both the To and From headers MUST be the address of the user account.
- The Autocrypt Setup Message MUST contain an Autocrypt-Setup-Message: v1 header.
- The Autocrypt Setup Message MUST have a multipart/mixed structure, and it MUST have as first part a human-readable description about the purpose of the message (e.g. text/plain or text/html or multipart/alternative).
- The second mime part of the message MUST have Content-Type application/autocrypt-setup, and SHOULD have Content-Disposition of attachment. Its content consists of the user’s ASCII-armored secret key, encrypted within an ASCII-armored OpenPGP symmetrically-encrypted message. Specifically, this means a block delimited with -----BEGIN PGP MESSAGE------ and -----END PGP
MESSAGE-----, which contains two OpenPGP packets: a Symmetric-Key Encrypted Session Key\textsuperscript{16} followed by a Symmetrically Encrypted Integrity Protected Data Packet\textsuperscript{17}.

- There MAY be text above or below the ASCII-armored encrypted data in the second MIME part, which MUST be ignored while processing. This allows implementations to optionally add another human-readable explanation.
- The encrypted payload MUST begin with an ASCII-armored RFC 4880 Transferable Secret Key\textsuperscript{18}. All trailing data after the first ASCII-armor ending delimiter MUST be stripped before processing the secret key. The ASCII-armored secret key SHOULD have an Autocrypt-Prefer-Encrypt header that contains the current accounts[addr].prefer_encrypt setting.
- The symmetric encryption algorithm used MUST be AES-128 or AES-256. The passphrase MUST be the Setup Code (see below), used with OpenPGP’s salted+iterated S2K algorithm\textsuperscript{19}.

### 5.4.2 Setup Code

The Setup Code MUST be generated by the implementation itself using a Cryptographically secure pseudorandom number generator (CSPRNG)\textsuperscript{20}, and presented directly to the user for safekeeping. It MUST NOT be included in the cleartext of the Autocrypt Setup Message, or otherwise transmitted over e-mail.

An Autocrypt Level 1 MUA MUST generate a Setup Code as UTF-8 string of 36 numeric characters, divided into nine blocks of four, separated by dashes. The dashes are part of the secret code and there are no spaces. This format holds about 119 bits of entropy. It is designed to be unambiguous, pronounceable, script-independent (Chinese, Cyrillic etc.), easily input on a mobile device and split into blocks that are easily kept in short term memory. For instance:

```
9503-1923-2307-
1980-7833-0983-
1998-7562-1111
```

An Autocrypt Setup Message that uses this structure for its Setup Code SHOULD include a Passphrase-Format header with value numeric9x4 in the ASCII-armored data. This allows providing a specialized input form during decryption, with greatly improved usability.

As a further measure to improve usability, it is RECOMMENDED to reveal the first two digits of the first block in a Passphrase-Begin header, sacrificing about 7 bits of entropy. Those digits can be pre-filled during decryption, which reassures the user that they have the correct code before typing the full 36 digits. It also helps mitigate a possible type of phishing attack that asks the user to input their Setup Code.

The headers might look like this:

```
Passphrase-Format: numeric9x4
Passphrase-Begin: 95
```

If those digits are included in the headers, they may also be used in the descriptive text that is part of the Setup Message, to distinguish different messages.

### 5.4.3 Setup Message Creation

An Autocrypt MUA MUST NOT create an Autocrypt Setup Message without explicit user interaction. When the user takes this action for a specific account, the MUA:

- Generates a Setup Code.

\textsuperscript{16} https://tools.ietf.org/html/rfc4880.html#section-5.3
\textsuperscript{17} https://tools.ietf.org/html/rfc4880.html#section-5.13
\textsuperscript{18} https://tools.ietf.org/html/rfc4880.html#section-11.2
\textsuperscript{19} https://tools.ietf.org/html/rfc4880.html#section-3.7.1.3
\textsuperscript{20} https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator
• Optionally, displays the Setup Code to the user, prompts the user to write it down, and then hides it and asks the user to re-enter it before continuing. This minor annoyance is a recommended defense against worse annoyance: it ensures that the code was actually written down and the Autocrypt Setup Message is not rendered useless.

• Produces an ASCII-armored, minimized OpenPGP Transferable Secret Key\textsuperscript{21} out of the key associated with that account.

• Symmetrically encrypts the OpenPGP transferable secret key using the Setup Code as the passphrase.

• Composes a new self-addressed e-mail message that contains the payload as a MIME part with the appropriate Content-Type and other headers.

• Sends the generated e-mail message to its own account.

• Suggests to the user to either back up the message or to import it from another Autocrypt-capable MUA.

A Level 1 MUA MUST be able to create an Autocrypt Setup Message, to preserve users’ ability to recover from disaster, and to choose to use a different Autocrypt-capable MUA in the future.

5.4.4 Setup Message Import

An Autocrypt-capable MUA SHOULD support the ability to find and import an Autocrypt Setup Message when the user has not yet configured Autocrypt (that is, when accounts[addr].secret_key is unset). An MUA in this state could look for such a message in several ways, including:

• If the user decides to enable Autocrypt for an account, and indicates to the MUA that an older MUA has already enabled Autocrypt on that account, the new MUA could ask the user to generate an Autocrypt Setup Message from the old MUA, and then wait (e.g., via IMAP IDLE\textsuperscript{22}) for such a message to arrive.

• The MUA could proactively scan the account’s mailbox for a message that matches these characteristics, and it could alert the user if it discovers one.

When looking for an Autocrypt Setup Message, the MUA may encounter messages that look similar to what it expects, but are not well-formed. If the MUA discovers an e-mail message that has the Autocrypt-Setup-Message header but its value is not v1, the MUA SHOULD ignore this message entirely.

When looking for an Autocrypt Setup Message, if the MUA discovers a message with the Autocrypt-Setup-Message: v1 header with To: and From: headers matching an account controlled by the MUA, but the message’s metadata and structure is not as expected, the MUA SHOULD alert the user that a malformed Setup Message has been found, and it SHOULD NOT offer to import the message.

If the MUA finds a good Autocrypt Setup Message, it should offer to import it to enable Autocrypt. If the user agrees to do so:

• The MUA prompts the user for their corresponding Setup Code. If there is a Passphrase-Format header in the outer OpenPGP armor and its value is numeric9x4, then the MUA MAY present a specialized input dialog assisting the user to enter a code in the format described above. If there is no Passphrase-Format header, or the value is unknown, then the MUA MUST provide a plain UTF-8 string text entry.

• The MUA should try decrypting the message with the supplied Setup Code. 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, \href{https://tools.ietf.org/html/rfc4880.html#section-11.2}{this is difficult to do correctly with GnuPG}\textsuperscript{23}.

• If it decrypts, then the MUA SHOULD update accounts[addr] according to the contents of the decrypted message, as discussed in Accounts controlled by the MUA (page 4).

\textsuperscript{21} https://tools.ietf.org/html/rfc4880.html#section-11.2
\textsuperscript{22} https://tools.ietf.org/html/rfc2177.html
\textsuperscript{23} https://dev.gnupg.org/T3277
See *Example Setup Message* (page 21).

Since Level 1 only recommends looking for a Setup Message when `accounts[addr].secret_key` is unset, some Level 1 MUAs might not look for or handle Setup Messages for an already-configured account at all. If two such MUAs share an account, and both MUAs have somehow enabled Autocrypt on it independently without discovery of a Setup Message, they will have different secret keys. This situation is bad because it may lead to intermittently unreadable mail on either or both MUAs.

These simple implementations can both keep Autocrypt enabled and avoid new unreadable mail if the user manually synchronizes secret keys. To do this, the user must first destroy their local secret key (page 17) on one MUA. Afterwards, that MUA can begin looking for a Setup Message again. A more sophisticated implementation may offer a more user-friendly way to detect this situation and resolve it.

## 6 User Interface

Ideally, Autocrypt users see very little UI. However, some UI is inevitable if we want users to be able to interoperate with existing, non-Autocrypt users.

### 6.1 Message Composition

If an MUA is willing to compose encrypted mail, it SHOULD include some UI mechanism at message composition time for the user to choose between encrypted message or cleartext. This may be as simple as a single checkbox.

If the Autocrypt recommendation is disable for a given message, the MUA MAY choose to avoid exposing this UI during message composition at all.

If the Autocrypt recommendation is either available or encrypt, the MUA SHOULD expose this UI with the recommended default (page 8) during message composition to allow the user to make a different decision.

If the Autocrypt recommendation is discourage, then the MUA SHOULD expose the UI in an inactive state. But if the user chooses to activate it (e.g., clicking on the checkbox), then the UI should display a warning to the user and ask them to confirm the choice to encrypt.

### 6.2 Account Preferences

Level 1 MUAs SHOULD allow the user to disable Autocrypt completely for each account they control (that is, to set `accounts[addr].enabled` to false). For level 1, we expect most MUAs to have Autocrypt disabled by default. See *Disabling Autocrypt* (page 17) for more details.

### 6.3 Helping Users get Started

This section provides recommendations for MUA implementations to help users start Autocrypt immediately after an account (with the address `addr`) was set up.

The MUA SHOULD scan the mailbox for messages sent by the user (wherever the messages might be) that show evidence of OpenPGP or Autocrypt usage. It is likely sufficient to only scan the messages sent during the last 30 days, as it is unlikely that the user used Autocrypt or OpenPGP actively if no such message was sent in the recent past.

From the set of all found sent messages, the MUA should determine the best action to take from the following list of choices. Earlier choices are better than later ones.

1. If an Autocrypt Setup Message was found:

   Start a setup process suggesting the user to import the setup message. If multiple Autocrypt Setup Messages are found, the most recent message should be preferred.
2. If a sent message with an Autocrypt header was found:
   Provide guidance for creating an Autocrypt Setup Message on the MUA that created the message.

3. If there is evidence of actively used OpenPGP software (for example if a secret key is available, some specific software is installed, etc.) or if encrypted mails are found:
   Inform the user about Autocrypt on <https://autocrypt.org/pgp-users>.

4. If no evidence for Autocrypt was found:
   Create a key with default settings and without a password in the background. Set your accounts[addr].prefer_encrypt to nopreference and start sending Autocrypt headers.

6.4 Disabling Autocrypt

Once Autocrypt is enabled for a given account (accounts[addr].enabled is set to true), the user might choose to disable it. By default, disabling should only set accounts[addr].enabled to false, and it SHOULD NOT destroy accounts[addr].secret_key. This preserves the user’s ability to read old encrypted e-mails, as well as being able to read encrypted e-mails that arrive after the user has disabled Autocrypt.

The act of re-enabling Autocrypt after it was disabled SHOULD leave accounts[addr].secret_key and accounts[addr].public_key intact, so that the user continues using the same key.

6.5 Destroying Secret Key Material

When disabling Autocrypt for an account, a Level 1 MUA MAY offer the user an opportunity to also destroy the secret key material for that account. Since Autocrypt clients generally do not discuss secret keys with users, a MUA offering this choice should use a phrase like “destroy access to encrypted messages”, rather than referring to “keys” or “key material”.

A MUA that allows the user this opportunity SHOULD clearly indicate to the user that the destruction of this secret key material will leave them unable to read any new messages that arrive encrypted. A MUA that only retains the encrypted form of archived messages SHOULD also indicate to the user that previously-received encrypted messages will become unreadable as well. Note that for some users, this is a desirable feature: “destroy all messages” is an appropriate action to take in some circumstances.

If the user selects this option, the MUA MUST clear both accounts[addr].secret_key and accounts[addr].public_key.

7 Appendix

7.1 E-mail Address Canonicalization

To keep consistent state referring to different but practically equivalent writings of an e-mail address, a MUA SHOULD canonicalize e-mail addresses when comparing them (for example for using an e-mail address as an index key).

Canonicalizing the domain part (the part after the @): A MUA SHOULD canonicalize the domain part using IDNA2008 Punycode conversion to ASCII24.

Canonicalizing the local part (the part before the @): Autocrypt-capable MUAs that encounter a peer’s e-mail address where the local part appears to be valid UTF-8 SHOULD canonicalize the local part by making it all lower-case using the “empty” locale (see W3C’s discussion on Case folding25 for more details).

SMTP specifications26 say the local part is technically domain-specific, and byte-for-byte arbitrarily sensitive. In practice, nearly every e-mail domain treats the local part of the address as a case-insensitive string. That is,
while it is permitted by the standards, John@example.org is very unlikely to deliver to a different mailbox than john@example.org.

An Autocrypt-capable MUA that is configured to use an account that has an e-mail address whose local part is not a valid UTF-8 string, or who cannot receive mail at the canonicalized form of their associated address SHOULD NOT enable Autocrypt on that e-mail account without an additional warning to the user.

Other canonicalization efforts are considered for later specification versions.

### 7.2 Example Autocrypt headers

Alice sends Bob a simple, unencrypted e-mail message that lets Bob write back encrypted if Bob is using an Autocrypt-enabled MUA:

```plaintext
Delivered-To: <bob@autocrypt.example>
From: Alice <alice@autocrypt.example>
To: Bob <bob@autocrypt.example>
Subject: an Autocrypt header example using Ed25519+Cv25519 key
Autocrypt: addr=alice@autocrypt.example; prefer-encrypt=mutual; keydata=
77x544mDMEXEcE6RYJKwYBBAhAw88AQdArjWkw3FAgiyiFbFBK4TzXcVBqPTB3gmz1C/UB70u120F2F
saMN1QGFldG9jcnlwcD5JeGFrGx1iYIEEYXJyAD4WlQTrhbtfozp14V6UmPyMVMUOTfjfgUCXEcE6Q1bDAAKCRDYMVUMT0
fjJlnQAOQFHUs6TicrxnTnETZfjUfM10PJDng/cD4xN80fnsQfEA22kr7VkcjeAE0C8VSTeV+
QFsmz551/lnWkwWkmnV0gE=
Date: Tue, 22 Jan 2019 12:56:25 +0100
Message-ID: <abe640bb-018d-4f9d-b4d8-1636d6164e22@autocrypt.example>
MIME-Version: 1.0
Content-Type: text/plain
This is an example e-mail with Autocrypt header and Ed25519+Cv25519 key (key fingerprint: ) as defined in Level 1 revision 1.1.
```

### 7.3 Example Autocrypt Gossip headers

After having received messages with Autocrypt headers from both Bob and Carol, Alice sends an e-mail to the two of them, with Autocrypt Gossip headers.

```plaintext
Delivered-To: <bob@autocrypt.example>, <carol@autocrypt.example>
From: Alice <alice@autocrypt.example>
To: Bob <bob@autocrypt.example>, Carol <carol@autocrypt.example>
Subject: an Autocrypt Gossip header example
Autocrypt: addr=alice@autocrypt.example; prefer-encrypt=mutual; keydata=
77x239mDMEXEcE6RYJKwYBBAhAw88AQdArjWkw3FAgiyiFbFBK4TzXcVBqPTB3gmz1C/UB70u120F2F
saMN1QGFldG9jcnlwcD5JeGFrGx1iYIEEYXJyAD4WlQTrhbtfozp14V6UmPyMVMUOTfjfgUCXEcE6Q1bDAAKCRDYMVUMT0
fjJlnQAOQFHUs6TicrxnTnETZfjUfM10PJDng/cD4xN80fnsQfEA22kr7VkcjeAE0C8VSTeV+
QFsmz551/lnWkwWkmnV0gE=
Date: Tue, 22 Jan 2019 12:56:29 +0100
Message-ID: <3ea1894a-a1cb-4e75-968c-7c1140117e3d@autocrypt.example>
MIME-Version: 1.0
Content-Type: multipart/encrypted; protocol="application/pgp-encrypted";
boundary="Pldq3hBodBceDdiav0rBQeh0u8JfdUHL"
```
Since Alice encrypts messages to herself, the above message can be decrypted by her private key as well (see the Example Setup Message (page 21) for access to her private key)

When decrypted, the encrypted part contains:

```
Autocrypt-Gossip: addr=bob@autocrypt.example; keydata=
```

(continues on next page)
Hi Bob and Carol,

I wanted to introduce the two of you to each other.

I hope you are both doing well! You can now both "reply all" here, and the thread will remain encrypted.

Regards,

Alice

---

7.4 Example Copy when a Reply can't be Encrypted

The message this is a reply to was sent encrypted, but this reply is unencrypted because I don't yet know how to encrypt to `bob@example.com`. If `bob@example.com` would reply here, my future messages in this thread will be encrypted.

7.5 Example User Interaction for Setup Message Creation

The Setup Code shown in this example can be used with Example Setup Message (page 21) below.

You'll need to use this Setup Code in your other e-mail app to use the Autocrypt Setup Message:

1742-0185-6197-
1303-7016-8412-
3581-4441-0597

7.6 Example User Interaction for Setup Message Receipt

To initiate the import of the Autocrypt Setup Message, the MUA can display a message like the example below:

ExampleMail has detected an Autocrypt Setup Message created by one of the other apps you use to access "alice@autocrypt.example". By importing the settings from this message, you can start using Autocrypt here in ExampleMail too!

Please enter the Setup Code displayed by your other e-mail app to proceed:

```
17  -  -  -  -  -
    -  -  -  -
```

(continues on next page)
7.7 Example Setup Message

Alice’s MUA sends her a Setup Message after showing her a Setup Code (the code used here is the one from Example User Interaction for Setup Message Creation (page 20)). The generated message looks like this:

```
To: alice@autocrypt.example
From: alice@autocrypt.example
Autocrypt-Setup-Message: v1
Subject: Autocrypt Setup Message
Date: Tue, 22 Jan 2019 12:56:29 +0100
Content-type: multipart/mixed; boundary="Y6fyGi9SoGeH8WwRaEdC6bbBcYOedDzrQ"

--Y6fyGi9SoGeH8WwRaEdC6bbBcYOedDzrQ
Content-Type: text/plain
This message contains all information to transfer your Autocrypt settings along with your secret key securely from your original device.

To set up your new device for Autocrypt, please follow the instructions that should be presented by your new device.

You can keep this message and use it as a backup for your secret key. If you want to do this, you should write down the Setup Code and store it securely.

--Y6fyGi9SoGeH8WwRaEdC6bbBcYOedDzrQ
Content-Type: application/autocrypt-setup
Content-Disposition: attachment; filename="autocrypt-setup-message.html"

<html><body>
<p>
This is the Autocrypt setup file used to transfer settings and keys between clients. You can decrypt it using the Setup Code presented on your old device, and then import the contained key into your keyring.
</p>
</body></html>
```

(continues on next page)
When decrypted with the Setup Code, the encrypted blob at the end contains:

```
-----BEGIN PGP PRIVATE KEY BLOCK-----
Autocrypt-Prefer-Encrypt: mutual
------
1FgEXEcE6RYJwYBAHaRw8BAQ0ArjWw3FAqyiFbFBKT4TzXcVBqPTB3gmz1C/U
b701u10AAP9XBe61zGOLx7zHH9ssUDEUt2pqgYGMzd0P3ulJ2AfvQ4RtBdhbg1j
ZUBhdXRvY3J5cHQsU2XhhbXbsZyiWBMNCAA+F1E64N7X6MdeF1eE5j8jFVD9E9H
444FAlxHBOkCgwMCQPCz2ACFwcKBwIGFQoJCaSBBYCAwEChgECF4AACgkQ8jFy
DE9H445kiwD/X65y08AsbwPhx2Z6zxGFCFcZc180uRouYDE6ad+yu944A/1WymiZM
JPyruvPTmB+Cu6uuy25XmiFkV/PPG2nFyWcMnF0EXEcE6RIKkwYBBAXVQVEFAQEH
QEL/B1tq0k84KmlqWZ2diKVrQrMttN8d7Bfprn4iAWEi6wAA/3/xFGP6UI7r
hTqg+07gmeEvAFYkfxRBR6sagAYI6W6TmfpQEk6IeAQYFggA1BYhBoFu1+jOnXhXpR
y/IxVQxPR+O0O3QcRWhTpsnLMAA0JEPIXVQxPR+O0WdABAMUdS2pHzGs10RknNQ
WduzQ8nUed9Njbe3zR+yfRAQDb5vQyvtWQKN4AQLTvxJN5X5AybPnn+WelATBha
Ga86AQ==
=BdSF
-----END PGP PRIVATE KEY BLOCK-----
```

### 7.8 Example Stored Draft

Alice composes a message to Bob, and manually chooses to encrypt in her MUA's UI. She does not send the message immediately, but stores it in a drafts folder. The message is encrypted (but not signed), and its encryption-related state is saved in the Autocrypt-Draft-Header (see **Storing Draft State** (page 11)). The encrypted payload also contains Bob’s key as an Autocrypt-Gossip header, to ensure encryption is possible when the message is picked up.

```
From: Alice <alice@autocrypt.example>
To: Bob <bob@autocrypt.example>
Subject: an example of a Draft
Date: Wed, 30 Jan 2019 18:48:38 +0100
Message-ID: <1b6828d5-61d5-40b4-8b42-bc318cfd2ad9@autocrypt.example>
MIME-Version: 1.0
Autocrypt-Draft-State: encrypt=yes; _by-choice=yes;
Content-Type: multipart/encrypted;
   protocol="application/pgp-encrypted";
   boundary="PLdq3hBodDceBdiavo4rbQeh0u8JfdUHL"

--PLdq3hBodDceBdiavo4rbQeh0u8JfdUHL
Content-Type: application/pgp-encrypted
Content-Description: PGP/MIME version identification
Version: 1

--PLdq3hBodDceBdiavo4rbQeh0u8JfdUHL
Content-Type: application/octet-stream; name="encrypted.asc"
Content-Description: OpenPGP encrypted message
Content-Disposition: inline; filename="encrypted.asc"

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

hF4DR2b2udXyHrYSAQdAqrEPggVyWpa80vaiGxNRno4EB+7XP5lkMxAEFNmhMw
0eWMEd8jzKpQpjs2aJrBr4bsaNxl1MFR2YmbPRQi0W+1u56oaddP10MFGAPT6QC
(continues on next page)

Autocrypt Specification 1.1.0
The encrypted payload contains:

Autocrypt-Gossip: addr=bob@autocrypt.example; keydata=

Content-Type: text/plain

Hi Bob,

this is a message where I'm not sure yet what to write. I'll just store it as a draft for now!

My other Autocrypt-enabled MUA should be able to pick this message up and see from the Autocrypt-Draft-State header that I chose to encrypt here. It will also have the key I would have used to encrypt if I had sent the message, from the contained Autocrypt-Gossip header.

Regards,

Alice

7.9 Document History

This document is kept under revision control. For detailed history, please consult the git logs. This section provides a high-level overview of what changed between revisions.

version 1.1

- change required algorithms and recommendation for key generation to Ed25519+Cv25519
- add Autocrypt-Draft-State header to preserve encryption state of drafts

version 1.0.1

27 https://github.com/autocrypt/autocrypt
• added Terminology section
• added Document History section
• specify how to deal with using non-Autocrypt keys (stripping excess user IDs)
• minor language, markup, and orthography cleanup

version 1.0.0
• first complete specification