Autocrypt Documentation

Release 0.5

hpk, dkg etc.al

Apr 11, 2017

Contents

The following in-progress documents are written for early adopters and contributors, MUA developers and privacy enthusiasts.

Introducing Autocrypt: E-Mail Encryption for Everyone introduction to why Autocrypt exists and why it is trying to achieve more e-mail encryption.

Autocrypt features discusses how the Autocrypt efforts is different from past e2e encryption efforts.

Example Data Flows and State Transitions Example data flows and MUA state transitions. This may be the easiest place to get started with the concrete ideas behind Autocrypt.

Guidance for Implementers of Autocrypt Level 0 Minimum requirements and implementer guidance for Level 0 Autocrypt-capable MUAs.

Interoperability With Other Cryptographic E-mail Mechanisms Guidance for integrating Autocrypt with other e-mail encryption mechanisms and UI for existing MUAs.

Future Enhancements to Autocrypt Future improvements for Autocrypt, beyond Level 0.

Potential ecosystem dangers of Autocrypt Some documented risks and dangers to the mail ecosystem, related to Autocrypt.

Frequently Asked Questions about Autocrypt

Glossary

CHAPTER 1

Introducing Autocrypt: E-Mail Encryption for Everyone

If users ask how they can secure their e-mail the answer should be as simple as: use an Autocrypt-enabled mail app!

Why improve e-mail? E-Mail has been declared dead many times but refuses to die. It remains the largest open federated identity and messaging eco-system, anchors the web, mobiles and continues to relay sensitive information between citizens and organisations. It has problems but do you prefer the proprietary, easy-to-track mobile phone number system to become the single source of digital identification?

Why a new approach to e-mail encryption? Encrypted e-mail has been around for decades, but has failed to see wide adoption outside of specialist communities, in large part because of difficulties with user experience and certification models. Autocrypt first aims to provide convenient encryption that is neither perfect nor as secure as traditional e-mail encryption, but is convenient enough for much wider adoption.

The social Autocrypt approach

The Autocrypt project is driven by a diverse group mail of app developers, hackers and researchers who are willing to take fresh approaches, learn from past mistakes, and collectively aim to increase the overall encryption of e-mail in the net. The group effort was born and named "Autocrypt" on December 17th 2016 by ~20 people during a 5-day meeting at the OnionSpace in Berlin. It's a dynamic, fun process which is open to new people, influences and contributions. See *contact channels and upcoming events* on how you may talk with us and who "we" are currently.

The technical Autocrypt approach

Autocrypt uses regular e-mail messages between people to piggyback necessary information to allow encrypting subsequent messages. Under the hood, Autocrypt uses e-mail headers for this information transfer. By default, no key management is visible to users. See *Autocrypt features* for more technical and UI cornerstones.

We are establishing this approach step-by-step using different "Levels" of implementation compliance. We are currently documenting *Level 0*, which we aim to see supported in Spring of 2017 by various mailers.

See *Current docs (work-in-progress)* for an index of all docs and discussion results so far.

CHAPTER 2

Autocrypt features

End-to-end encrypted e-mail has been around for decades, but has failed to see wide adoption outside of specialist communities, in large part because of difficulties with user experience and certification models. To better understand how the Autocrypt effort is different from previous ones here are some of its features:

- **Protect first against passive data-collecting adversaries**, resist the temptation to early-add complexity which aim to prevent active attacks. See RFC7435 A New Perspective for some motivation of this and the next points.
- **Focus on incremental deployment**, always consider that there will be both Autocrypt-enabled mail apps and traditional plain ones, interacting with each other.
- **Don't ask users anything about keys, ever.** And minimize and usability-test what needs to be decided by users and include resulting UI guidance in the specs. Minimize friction for people using multiple mail apps with their accounts.
- **Go for mail app changes only**, don't require changes from mail providers or depend on third party services, allowing fluid development of deployable code and specs.
- **Use decentralized, in-band key discovery.** Make mail apps tell each other how and when to encrypt to each other by attaching neccessary information along with mails.
- **Implement and specify "Level 0" support in several mail apps in spring 2017.** Keep Level 0 minimal enough that it's easy for developers to adopt it and we can start to drive efforts from real-life experiences. Please see *Guidance for Implementers of Autocrypt Level 0*.

CHAPTER 3

Channels

If you want to help, including offering constructive criticism, you may:

- join the Autocrypt mailing list
- join chats at **#autocrypt on freenode or matrix.org**.
- collaborate through PRs, issues and edits on our github Autocrypt repo

Currently involved are developers from K9/Android, OpenKeyChain/Android, Enigmail, Mailpile, Bitmask/LEAP as well as people from the nextleap, enzevalos research projects and the ACLU.

CHAPTER 4

Upcoming events

- Dec 2016: at 33c3, Hamburg, scheduled talk at the We Fix the Net session and probably another separate one.
- Jan 2017: a prospective lightning talk from dkg at RealWorldCrypto 2017 in New york
- Mar 2017: Autocrypt sessions at the Internet Freedom Festival with hackers and users, several Autocrypt-people there.
- April/May 2017: next Autocrypt unconf-hackathon planned roughly around DE/NL/CH

CHAPTER 5

Example Data Flows and State Transitions

Autocrypt key discovery happens through headers of mail messages sent between mail apps. Similar to TLS's machine to machine handshake, users first need to have a cleartext mail exchange. Subsequent mails from the receiving peer will may then be encrypted. Mail apps show encryptability to their users at "compose-mail" time and give them a choice of encryption or cleartext, defaulting to what the other side has specified in their header.

These examples try to walk a new reader through the basic flow.

Note: Autocrypt key discovery is safe only against passive eavesdroppers. It is trivial for providers to perform active downgrade or man-in-the-middle attacks on Autocrypt's key discovery. Users may, however, detect such tampering if they verify their keys out-of-band at some later point in time. We hope this possiblity will keep most providers honest or at least prevent them from performing active attacks on a massive scale.

Please also see https://github.com/autocrypt/autocrypt/tree/master/src/tests/data for specific examples of Autocrypt messages.

Contents

Basic network protocol flow

Establishing encryption happens as a side effect when people send each other mail:

- A MUA (mail user agent) always adds an `Autocrypt:` header to all messages it sends out.

  The autocrypt header contains all necessary information to allow encryption (especially the key; see *Deriving a Parsed Autocrypt Header from a Message* for the format in detail).

- A MUA will scan incoming mails for encryption headers and associate the info with a canonicalized version of the `From:` address contained in the **RFC 822** message.
- A MUA will encrypt a message if it earlier saw encryption keys (and the request to encrypt) for all recipients.

"Happy path" example: 1:1 communication

Consider a blank state and a first outgoing message from Alice to Bob:

```
From: alice@a.example
To: bob@b.example
...
```

Upon sending this mail, Alice's MUA will add a header which contains her encryption key:

```
Autocrypt: to=alice@a.example; type=p; prefer-encrypted=yes; key=...
```

Bob's MUA will scan the incoming mail, find Alice's key and store it associated to the `alice@a.example` address taken from the `to`-attribute. When Bob now composes a mail to Alice his MUA will find the key and signal to Bob that the mail will be encrypted and after finalization of the mail encrypt it. Moreover, Bob's MUA will add its own Encryption Info:

```
Autocrypt: to=bob@b.example; type=p; prefer-encrypted=yes; key=...
```

When Alice's MUA now scans the incoming mail from Bob it will store Bob's key and the fact that Bob sent an encrypted mail. Subsequently both Alice and Bob will have their MUAs encrypt mails to each other.

If `prefer-encrypted` is sent as 'yes' the MUA MUST default to encrypting the next e-mail. If it is set as 'no' the MUA MUST default to plaintext. If `prefer-encrypted` is not sent the MUA should stick to what it was doing before. If the attribute has never been sent it's up to the MUA to decide. The save way to go about it is to default to plaintext to make sure the recipient can read the e-mail.

We encourage MUA developers to propose heuristics for handling the undirected case. We will document the best approaches to develop a shared understanding.

Group mail communication (1:N)

Consider a blank state and a first outgoing message from Alice to Bob and Carol. Alice's MUA add a header just like in the 1:1 case so that Bob and Carol's MUA will learn Alice's key. After Bob and Carol have each replied once, all MUAs will have appropriate keys for encrypting the group communication.

It is possible that an encrypted mail is replied to in cleartext (unencrypted). For example, consider this mail flow:

```
Alice -> Bob, Carol
Bob -> Alice, Carol
Carol -> Alice  (not to Bob!)
```

Alice and Carol have now all encryption keys but Bob only has Alice's because he never saw a mail from Carol. Alice can now send an encrypted mail to Bob and Carol but Bub will not be able to respond encrypted before his MUA has seen a mail from Carol. This is fine because Autocrypt is about **opportunistic** encryption, i.e. encrypt if possible and otherwise don't get in the way of users.

Losing access to decryption key

If Alice loses access to her decryption secret:

- she lets her MUA generate a new key
- her MUA will add an Encryption-Info header containing the new key with each mail
- receiving MUAs will replace the old key with the new key

Meanwhile, if Bob sends Alice a mail encrypted to the old key she will not be able tor ead it. After she responds (e.g. with "Hey, can't read your mail") Bob's MUA will see the new key and subsequently use it.

Todo

Check if we can encrypt a mime mail such that non-decrypt-capable clients will show a message that helps Alice to reply in the suggested way. We don't want people to read handbooks before using Autocrypt so any guidance we can "automatically" provide in case of errors is good.

Note: Unless we can get perfect recoverability (also for device loss etc.) we will always have to consider this "fatal" case of losing a secret key and how users can deal with it. Especially in the federated e-mail context We do not think perfect recoverability is feasible.

Downgrading / switch to a MUA without Autocrypt support

Alice might decide to switch to a different MUA which does not support Autocrypt.

A MUA which previously saw an Autocrypt header and/or encryption from Alice now sees an unencrypted mail from Alice and no encryption header. This will disable encryption to Alice for subsequent mails.

CHAPTER 6

Guidance for Implementers of Autocrypt Level 0

This document describes the basic capabilities required for a MUA to be Autocrypt-capable at Level 0. Some Autocrypt-capable clients may choose to go beyond these features, and future Levels of Autocrypt may require more specific control.

Throughout this document, we refer to a Mail User Agent (MUA) as though it was only capable of controlling a single e-mail account. A MUA that is capable of connecting to multiple e-mail accounts should have a separate Autocrypt state for each e-mail account it has access to.

Contents

Requirements on MUA/E-mail Provider interactions

Autocrypt tries to impose minimal requirements on how MUAs and e-mail services interact. We assume that an Autocrypt-capable MUA has credentials and capabilities to perform these network services:

- The ability to send e-mail (e.g. via SMTP or Submission) where the MUA can control the entire message being sent, including both message headers and message body.
- The ability to receive e-mail where the MUA gets access to the entire message being received, including both message headers and message body.
- Access to a special (IMAP) Shared Message Archive (SMA) folder which can be accessed by all MUAs of a user's devices to co-ordinate between them. In Level 0 this is only used for ensuring that only one MUA has Autocrypt enabled for an e-mail account at once.

If a particular e-mail account does not expose these features (e.g. if it only exposes a javascript-driven web interface for message composition that does not allow setting of e-mail headers, or if it only offers POP access to the incoming mail) 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.

Todo

Discuss with webmail developers how to work with, refine the interactions.

Secret key generation and storage

The MUA MUST be capable of generating and storing two RSA 2048-bit secret keys, one for signing and self-certification and the other for decrypting. It MUST be capable of assembling these keys into an OpenPGP certificate (RFC 4880 "Transferable Public Key") that indicates these capabilities.

These secret keys MUST 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. These secret keys MUST never be sent over the wire to any other party.

When an Autocrypt-enabled MUA configures an e-mail account, it should generate these keys and store them locally. Then it should proceed to try to "claim" the account to lock out other MUAs of the same users. In Level 0 only one MUA can send and receive encrypted mail through Autocrypt mechanisms.

Claiming the Account

Only one Level 0 MUA can have Autocrypt enabled for a given account at a time. The Autocrypt-enabled MUA "claims" the account so that others disable their Autocrypt features.

The Shared Mail Archive MUST contain a named location mechanism that all other Autocrypt clients can see. For example, an IMAP mailbox would have a named folder. Autocrypt uses the special name *_autocrypt_sma* to store "claim" announcements.

The MUA looks in the special location for a message whose form matches the standard claim announcement and is valid. If such a message is present, the MUA disables its Autocrypt features for this account.

If the special location does not exist, or it exists, but there are no valid claim announcements in it, the MUA crafts its own claim announcement and places it in the special location.

Todo

- Document the claim announcement format
- Clarify concerns about race conditions, case-sensitivity, etc.

Header injection in outbound mail

During message composition where the message will be marked as `From:` an e-mail address that the Autocrypt-capable agent knows the secret key material for, it should always include an Autocrypt header with the associated public key material as the `key=` attribute, and it should include the `to=` attribute for recipients to match on. The most minimal Level 0 MUA will only include these two attributes.

If the `From:` address changes during message composition (e.g. if the user selects a different outbound identity, the Autocrypt-capable client MUST change the `Autocrypt:` header.

See *"Happy path" example: 1:1 communication* for examples of outbound headers and the following sections for header format definitions and parsing.

Deriving a Parsed Autocrypt Header from a Message

The `Autocrypt:` header MUST have the following format:

```
Autocrypt: to=a@b.example.org; [type=(p|_*);] [prefer-encrypted=(yes|no);] key=BASE64
```

Where the value of `key` includes a Base64 representation of a minimal key. For now we only support `p` as the type, which represents a specific subset of OpenPGP (see the the next section). `key` MUST be the last attribute.

`prefer-encrypted` indicates that agents should default to encrypting when composing e-mails to this recipient. If `prefer-encrypted` is not set, the value of `prefer-encrypted` is `nopreference`. If `prefer-encrypted` is set, but neither `yes` nor `no`, the MUA must skip the header as invalid.

Additional attributes unspecified here are also possible before the `key` attribute. If a MUA encounters an unknown attribute, if the attribute name starts with an underscore (_), this is a "non-critical" attribute. The MUA MUST ignore the non-critical attribute and continue parsing the header. If the unknown attribute name does *not* start with an underscore, it is a "critical" attribute, and the MUA must skip the header as invalid.

If a Level 0 MUA encounters an otherwise-valid header which has `type` set to something other than `p` it MUST skip the header as invalid.

When parsing an incoming message, a Level 0 MUA MUST examine all headers with the name `Autocrypt:` and collect all valid headers in a list. If the list of valid headers has more than one entry, it should be treated as 0 valid headers (that is, it should return `null`).

`type=p`: OpenPGP Based key data

For maximum interoperability and sanity a certificate sent by an Autocrypt-enabled Level 0 MUA MUST send an OpenPGP "Transferable Public Key" (see RFC 4880 §11.1) containing exactly these five OpenPGP packets:

- a primary key `Kp`
- a uid that SHOULD be set to the e-mail address of the account
- a self signature
- an encryption-capable subkey `Ke`
- a binding signature over `Ke` by `Kp`

These packets should be assembled in binary format (not ASCII-armored), and then base64-encoded. During interpretation, whitespace should be stripped before base64-decoding.

A Level 0 MUA MUST be capable of processing and handling 2048-bit RSA keys. It SHOULD be capable of handling Curve 25519 keys (ed25519 for `Kp` and cv25519 for `Ke`), but some underlying toolkits may not yet support Curve 25519.

Internal state storage

Note: You should be familiar with *"Happy path" example: 1:1 communication* before reading the following.

If a remote peer disables Autocrypt or drops back to using a non-Autocrypt MUA only we must be able to disable sending encrypted mails to this peer automatically. MUAs capable of Autocrypt level 0 therefore MUST store state about the capabilities of their remote peers.

Agents MAY also store additional information gathered for heuristic purposes, or for other cryptographic schemes. However, in order to support future syncing of Autocrypt state between agents, it is critical that Autocrypt-capable agents maintain the state specified here.

Conceptually, we represent this state as a table named `autocrypt_peer_state` indexed by the peer's *canonicalized e-mail address* and key type. In level 0, there is only one type, `p`, so level 0 agents can implement this by indexing only the peer's e-mail address.

For each e-mail and type, an Agent MUST store the following attributes:

- `pah`: Parsed Autocrypt header, which could be `null`
- `changed`: UTC Timestamp when `pah` was last changed
- `last_seen`: Most recent UTC time that `pah` was confirmed

Autocrypt-compatible agents SHOULD track and store in `autocrypt_peer_state` a parsed interpretation `pah`, which is not necessarily the literal header emitted (for the literal header, see next section). The `pah` MUST contain the following fields:

- `key` – the raw key material, after base64 decoding
- `prefer_encrypted` – a tri-state: `nopreference`, `yes`, or `no`

Updating internal state upon message receipt

When first encountering an incoming e-mail `M` from an e-mail address `A`, the MUA should follow the following `autocrypt_update` algorithm:

- Set a local `message_date` to the `Date:` header of `M`.
- If `message_date` is in the future, set `message_date` to the current time.

Todo

This implies that Autocrypt clients keep track of whether they have encountered a given message before, but does not provide them with guidance on how to do so. Message-ID? Digest of full message body? The consequences of re-triggering the message receipt process should only matter for messages that are erroneously marked with a future date. Another approach that would not require keeping track of the message would be to simply ignore messages whose `Date: header is in the future.

- Set a local `message_pah` to be the `Autocrypt:` header in `M`. This is either a single Parsed Autocrypt header, or `null`.
- If `message_pah` is `null`, and the MUA knows about additional OpenPGP keys, then we replace `message_pah` with a `synthesized_pah` generated from the message itself:
  - If the message is not cryptographically signed, or there is an unverifiable or invalid message signature, `synthesized_pah` is `null`.
  - Alternately, the message is cryptographically signed, and the signature is verified and comes from a known OpenPGP certificate `K`: If `K` is not encryption-capable (i.e. if the primary key has no encryption-capabilities marked, and no valid subkeys are encryption-capable), or if K does not have an OpenPGP User ID which contains the e-mail address in the message's `From:`, then `synthesized_pah` is also `null`. Otherwise, with an encryption-capable `K`, the `key` element of `synthesized_pah` is set to `K`. In this case, the `prefer_encrypted` element of `synthesized_pah` is set based on whether the message is also encrypted in addition to being signed. If the message is encrypted, then `prefer_encrypted` is set to `yes`. If it is not encrypted, then `prefer_encrypted` is set to `nopreference`.

Note: We do *not* synthesize the Autocrypt header from any `application/pgp-keys` message parts. This is because it's possible that an attached OpenPGP key is not intended to be the sender's OpenPGP key. For example, Alice might send Bob Carol's OpenPGP key in an attachment, but Bob should not interpret it as Carol's key.

Todo

Maybe move `synthesized_pah` into *Interoperability With Other Cryptographic E-mail Mechanisms* ?

- Note: The agent continues this message receipt process even when `message_pah` is `null`, since updating the stored state with `null` is sometimes the correct action.
- Next, the agent compares the `message_pah` with the `pah` stored in `autocrypt_peer_state[A]`.
- If `autocrypt_peer_state` has no record at all for address `A`, the MUA sets `autocrypt_peer_state[A]` such that `pah` is `message_pah` and `changed` and `last_seen` are both `message_date`, and then terminates this receipt process.

- If `autocrypt_peer_state[A]` has `last_seen` greater than or equal to `message_date`, then the agent stores `message_pah` and terminates this receipt process, since it already knows about something more recent. For example, this might be if mail is delivered out of order, or if an inbox is scanned from newest to oldest.
- If `autocrypt_peer_state[A]` has a `last_seen` less than `message_date`, then we compare `message_pah` with the `pah` currently stored in `autocrypt_peer_state[A]`.

  This is done as a literal comparison using only the `key` and `prefer_encrypt` fields, even if the Agent stores additional fields as an augmentation, as follows:

  - If `key` is bytewise different, or if `prefer_encrypted` has a different value, then this is an *update*.
  - If `key` and `prefer_encrypted` match exactly, then it is considered a *match*.
  - If both `pah` and `message_pah` are `null`, it is a *match*.
  - If one is `null` and the other is not `null`, it is a *update*.
- In the case of a **match**, set `autocrypt_peer_state[A].last_seen` to `message_date`.
- In the case of an **update**, set `autocrypt_peer_state[A].pah` to `message_pah` and `autocrypt_peer_state[A].last_seen` and `autocrypt_peer_state[A].changed` to `message_date`.

Note: The above algorithm results in a non-deterministic `autocrypt_peer_state` if two Autocrypt headers are processed using the same `message_date` (depending on which message is encountered first). For consistency and predictability across implementations, it would be better to have a strict ordering between parsed Autocrypt headers, and to always select the lower header in case of equal values of `message_date`.

Note: OpenPGP's composable certificate format suggests that there could be alternate ways to compare `key` values besides strict bytewise comparison. For example, this could be done by comparing only the fingerprint of the OpenPGP primary key instead of the keydata. However, this would miss updates of the encryption-capable subkey, or updates to the capabilities advertised in the OpenPGP self-signature. Alternately, the message receipt process could incorporate fancier date comparisons by integrating the timestamps within the OpenPGP messages during the date comparison step. For simplicity and ease of implementation, level 0 Autocrypt-capable agents are expected to avoid these approaches and to do full bytestring comparisons of `key` data instead.

Todo

the spec currently doesn't say how to integrate Autocrypt processing on message receipt with spam filtering. Should we say something about not doing Autocrypt processing on message receipt if the message is believed to be spam?

Provide a recommendation for message encryption

On message composition, an Autocrypt-capable agent also has an opportunity to decide whether to try to encrypt an e-mail. Autocrypt aims to provide a reasonable recommendation for the agent.

Any Autocrypt-capable agent may have other means for making this decision outside of Autocrypt (see *Interoperability With Other Cryptographic E-mail Mechanisms*). Autocrypt provides a recommendation to this process, but there is no requirement for Autocrypt-capable agents to always follow the Autocrypt recommendation.

That said, all Autocrypt-capable agents should be able to calculate the same Autocrypt recommendation due to their internal state.

The Autocrypt recommendation depends on the list of recipient addresses for the message being composed. When the user edits the list of recipients, the recommendation may change. The MUA should reflect this change.

Note: It's possible that the user manually overriddes the Autocrypt recommendation and then edits the list of recipients. The MUA SHOULD retain the user's manual choices for a given message even if the Autcrypt recommendation changes.

Todo

Discuss how to deal with the case where the user manually selects encryption and subsequently adds a recipient whom the MUA has no key.

Autocrypt can produce three possible recommendations to the agent during message composition:

- `disable`: Disable or hide any UI that would allow the user to choose to encrypt the message. Prepare the message in cleartext.
- `available`: Enable UI that would allow the user to choose to encrypt the message, but do not default to encryption. Prepare the message in cleartext.
- `encrypt` : Enable UI that would allow the user to choose to send the message in cleartext, and default to encryption. Prepare the message as an encrypted message.

Todo

The Autocrypt recommendation should probably change depending on whether the mail is a reply to an encrypted e-mail or not.

Recommendations for single-recipient messages

For level 0 MUAs, the Autocrypt recommendation for message composed to a single recipient with e-mail address `A` is derived from the value stored in `autocrypt_peer_state[A]`.

If the `pah` is `null`, or if `pah.key` is known to be unusable for encryption (e.g. it is otherwise known to be revoked or expired), then the recommendation is `disable`.

If the `pah` is not `null`, and `prefer-encrypted` is `yes`, then the recommendation is `encrypt`.

If `pah` is not `null`, and `prefer-encrypted` is either `no` or `nopreference`, then the recommendation is `available`.

Recommendations for messages to multiple addresses

For level 0 agents, the Autocrypt recommendation for a message composed to multiple recipients is derived from the recommendations for each recipient individually.

If any recipient has a recommendation of `disable` then the message recommendation is `disable`.

If every recipient other than "myself" (the e-mail address that the message is `From:`) has a recommendation of `encrypt` then the message recommendation is `encrypt`.

Otherwise, the message recommendation is `available`.

Encrypt outbound mail as requested

As the user composes mail, in some circumstances, the MUA may be instructed by the user to encrypt the message. If the recipient's keys are all of `type=p`, and the sender has keys for all recipients (as well as themselves), they should construct the encrypted message as a PGP/MIME (RFC 3156) encrypted+signed message, encrypted to all recipients and the public key whose secret is controlled by the MUA itself.

For messages that are going to be encrypted when sent, the MUA MUST NOT leak the cleartext of drafts or other partially-composed messages to the SMA (e.g. in the "Drafts" folder).

If there is any chance that the message could be encrypted, the MUA SHOULD encrypt drafts only to itself before storing in any Drafts folder on the SMA.

Specific User Interface Elements

Ideally, Autocrypt users see very little UI. They might never see any UI at all by default. However, some UI is inevitable, even if only tucked away in an arcane "preferences pane" or something.

Account Preferences

Level 0 MUAs MUST allow the user to disable Autocrypt completely for each account they control.

If Autocrypt is enabled for a given account, the MUA MUST allow the user to specify whether they explicitly prefer encryption for inbound messages, or explicitly prefer cleartext for inbound messages, or choose to express no preference. The default SHOULD be "no preference" unless the MUA has good reason to know better.

Please see *Example User Interface for Autocrypt* for specific examples of how this might look.

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 an 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 during message composition to allow the user to make a different decision.

CHAPTER 7

E-mail Address canonicalization

Domain part (the part after the `@`):

Todo

We need to choose a canonicalization form for the domain side of the e-mail address. There are risks for user presentation around phishing with IDNs, which we should be careful about.

Local part (the part before the `@`):

SMTP specs say this part is 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`. Autocrypt-aware MUAs will canonicalize the local part of an e-mail address by making it all lower-case.

Todo

some people (and some e-mail domains) have known variations which all deliver to the same account. For example, the mailbox that receives `john@example.org` might automatically receive all mail addressed like `john-whatever@example.org`. gmail today supports arbitrary dot injection (e.g. `johndoe@example.org` delivers to the same mailbox as `john.doe@example.org`). Do we want to try to support these somehow? It would be simplest to declare anyone using aliasing schemes like this as out-of-scope for Autocryptv1.

Todo

do we want to allow sophisticated users to explicitly merge known shared aliases as long as the domain side stays the same? For example, if i happen to know that `jdoe@example.org` delivers to the same mailbox as `john@example.org`, can i declare that to an Autocrypt-aware MUA? How would such an explicit merge affect state management?

CHAPTER 8

Potential ecosystem dangers of Autocrypt

This document is a place to describe particular concerns that Autocrypt creates for the e-mail ecosystem as a whole. It does not address attacks against the cryptography or compromises to the message confidentiality it aims to support.

These risks may not be large risks, or they may be mitigatable in some way, but we document them here for general awareness.

In all, we currently believe that the benefits to the ecosystem of having more end-to-end message confidentiality outweigh these potential risks.

Failures of Search

If Autocrypt clients are incapable of searching encrypted mail, users of Autocrypt-capable clients may find e-mail less useful for normal communication.

Message Deliverability

Autocrypt headers that use RSA 2048 are large enough that, when unwrapped, they exceed the SMTP line length limit of 1000 ASCII characters.

It's conceivable that some MTAs or MUAs will choke upon trying to deal with these headers, and render the message undeliverable or unreadable. We have no evidence of this happening today (December 2016), but maybe we're just not yet tickling the systems that have these problems.

Possible mitigations:

- sending duplicate headers each with parts of the key data. But this makes reassembly and message-parsing logic significantly more complex, and it would be nice to not need it.

Denial of Service: malicious creation of unreadable mail

An active attacker who wants to interrupt communication between two parties can do so if they know that one party uses an Autocrypt-capable agent. Consider the case where Mallory wants to interrupt communications between Alice and Bob, and she knows that Bob uses an Autocrypt-capable client.

Mallory crafts a new key K. She can throw away the secret key material entirely if she wants to. She then forges an e-mail from Alice and adds an Autocrypt header to it containing that public key and *prefer-encrypted=yes*. If Bob writes a message to Alice after receiving that key, and before receiving any other legitimate message to Alice, his message will be encrypted to a key that Alice cannot read.

this represents a risk to Alice, even if she has never adopted an Autocrypt-capable client in the first place.

Mitigations:

- Alice's next mail to Bob will correct Bob's client's state so that futre mails will be back to Alice's actualy preferred state. So the attacker must sustain a series of forgeries if the denial of service attack is intended to be sustained.
- we should specify that any spam/malware flag set from a filter that the user trusts should be sufficient to discourage processing of Autocrypt headers, so that Mallory needs to craft a sufficiently-plausible message (including DKIM and whatever other indicators the filters care about) to make it into the Autocrypt-capable agent's internal state storage.

Killing off strong encryption

Autocrypt is significantly weaker than traditional models of mail encryption. In particular, it provides no resistance to an active attacker (an attacker who can modify and/or inject mail as it passes through the SMTP network). The no-UI feature makes it so that most users will never properly verify each other's encryption keys.

There is a concern that if opportunistically-encrypted mail becomes the standard, no one will bother to implement good UX for users in strong identity verification.

Mitigations:

- make out-of-band verification of keys between users fun and thus increase the risk for attackers to get detected.
- research how "level 2" Autocrypt could evolve to offer automated support against active attackers.

A note on Autocrypt and provider spam/malware filters

Mike Hearn raised some fundamental concerns in his Modern anti-spam and E2E crypto post on the modern crypto mailing list on how end-to-end encrypted mails and spam infrastructure possibly interfere. While we may conceive new ways to fight spam in an E2E setting by increased DKIM usage and other additional measures the topic is a serious one as adoption of more encrypted mails could be seriously hampered if encryption can bypass current anti-spam technology.

Autocrypt works with existing provider Anti-Spam infrastructures because they can continue to check the initial clear-text mails for suspicious content. Only if a user replies to a (likely non-spam) mail will Autocrypt make a MUA send an encryption key. Without being able to get sufficiently many replies from users it will likely be to massively harvest encryption keys; there is no central registery for key-mail address relations. Massive collection of key/mailaddress associations would require co-operation from or compromise of big mail providers which is unlikely given they have been fighting unsolicited mails for decades and their business models depend on it.

CHAPTER 9

Interoperability With Other Cryptographic E-mail Mechanisms

Many MUAs that aim to become Autocrypt-compatible will already have implementations of other e-mail encryption mechanisms.

We have concrete guidance for those MUAs that we hope is useful.

Message encryption recommendations

An Autocrypt-capable agent that also incorporates the OpenPGP "Web of Trust" might already know about a non-Autocrypt public key that it considers to be correctly bound to the recipient e-mail address. It may wish to prefer such a key, and to decide to use for a given outbound message over any recommendations provided by Autocrypt.

For current OpenPGP users

- What about other keys, that i have been using with other properties? (smart-card, RSA, ...)
  - You can still create a compatible header with a tool we will provide. We are targeting users who have not used pgp before. Nevertheless most clients will still support other key formats. But they are not required to.

Todo

More guidance here!

Interoperability with existing PGP practises

should Autocrypt keys appear on key servers?

- no!

should i add rcvd Autocrypt keys into my PGP keyring? (if my mua already supports PGP)

- yes

should my own Autocrypt keys appear in my keyring?

- no (why not? how else can we do encrypt-to-self, or message signing?)

can I put my regular pgp keys into Autocrypt?

- MUAs should not provide UI for importing keys for Level 1
- allowed for Level 0 to get traction early on (as replacement for keyservers)

can I use someone's pgp key that i have for encrypting mail to that person?

- This would work like without Autocrypt

if i have for a person an non-Autocrypt pgp key and an Autocrypt key, which one do i use to encrypt mails for that person?

- Look up e-mail address in pgp keyring
- if there is a key that has better user ID validity for the matching address than "unknown", use that one
- else look up a key from the Autocrypt state (which is also in the keyring)

two target audiences:

- end-users
- mail software devs

CHAPTER 10

Frequently Asked Questions about Autocrypt

Why are you using headers rather than attached keys?

Attachments are visible to users of non Autocrypt-compatible MUAs, while headers are not. We don't want to present distracting or confusing material to those users.

Why are you sending keys in all the mails and not just announcing capabilities?

We did this in a previous version. We decided against it because it requires the MUA to keep the information who announced Autocrypt and who they requested keys from.

Why are we using IMAP folders rather than self send messages for multi device?

Self send messages end up in your inbox and might be confusing to users. They are likely also processed by your spam protection and might look like spam.

Why do you aim to use ed25519 - it's not supported by X?

They give us much smaller keys that lead to smaller headers and make it easier to include them. You can even write them down as a backup code. We want to support implementation where needed.

So you say you care about header size... but then you type out prefer-encrypt?

An ECC key is roughly 500 bytes formated in Base64 and RSA 2048 key is 1750 bytes. The Length of attribute name does not matter so much. So we opted for readability.

Why do you drop all headers if there is more than one?

Because of multi-agent usage we may have to handle an inconsistent stream of headers already. Making this an inconsistent stream of multiple keys with priorities sounds like a lot of pain.

What if I want two different keys announced?

If you really care about supporting other keys than what we use in Autocrypt there is the OpenPGP header that could use some standardization and automatic client support. Feel free to innovate there.

If we want to enable multiple headers in the future we can still add Autocrypt headers with a critical attribute 'priority'. Versions that do not support it yet will drop these headers and fall back to the one without priority.

Why do you use the `to=` attribute rather than the uid from the key?

We need to store state about the key to use for a given e-mail address. Just importing the key into a keyring won't cut it.

We want to be able to handle the header without having to parse the key first. We believe that using the 'to' attribute will be more forward compatible. For example we discussed hashing the uid in the keys so in case they leak to pgp keyservers they do not leak the e-mail address. This would not be compatible with requiring the e-mail address as the uid.

How does Autocrypt interact with message signing?

In general, Autocrypt assumes that mail is either plaintext mail, or it is both encrypted and signed. This assumption makes it possible to create a simpler user experience.

While there are valid usecases for signed, unencrypted mail, or for encrypted, unsigned mail, they are not the use case targeted by Autocrypt.

Why use OpenPGP and PGP/MIME instead of some other encryption tech?

We picked a commonly-understood and implemented mail encryption technology so that implementers wouldn't need to start from scratch.

Future levels of the Autocrypt specification may support different encryption technologies, but the main immediate goal is to get wider adoption, not to re-invent the encryption mechanism itself.

Please see *key-formats* for more discussion.

Why don't you use the `User-Agent` header to detect different mail apps?

Not all mail apps implement the `User-Agent` header (and there is an ongoing effort to discourage its use as a way to reduce metadata leakage). Also, some mail apps are used only to read mail, and are not used to send at all, so the remote peer can't see anything about those specific apps.

We could encourage each MUA to publish a UUID to inform the remote peer that multiple mail apps are in use, but it's not clear that this offers much benefit, and it leaks information that we don't need to leak.

What about spammers accidentally downgrading encryption?

A spammer who forges mail from a given address could potentially downgrade encryption for that person as a side effect. Please see *level0/public-key-management* for details about expected interaction with spam filters.

How does Autocrypt interact with today's mailing list managers?

Mailing lists that distribute cleartext (unencrypted) mail may end up distributing their user's public key material in the `Autocrypt:` headers of the distributed mail. For mailing lists that rewrite `From:` headers, these `Autocrypt:` headers will be dropped by recipients, which is fine.

For encrypted mailing lists like schleuder, we haven't done a full analysis yet. Suggestions welcome!

Why don't you encourage gossiping keys of other users?

This is a plausible future improvement for Autocrypt. But being willing to accept gossiped keys for other users presents a more complicated and risky public-key state management situation for the receiving client. For example, what if one client gets multiple different keys for a target address from different gossiping peers – should the client encrypt to all keys or just some? How should those keys interact with keys received from the end peer directly? Because of these complications, we're sidestepping this problem for level 0.

We welcome drafts proposing sensible ways to manage key gossip in group e-mail communication for future levels of Autocrypt.

Why can only one Level 0 MUA to "claim" an e-mail account for Autocrypt?

In the event that two Autocrypt-enabled agents operate a single e-mail account, they could clash and cause serious usability problems. In particular, if they each manage their own secret key material, communicating peers might arbitrarily choose one key or another to encrypt to, and then certain mails will be unreadable with certain agents, in an apparently-arbitrary pattern based on the origin of the remote peer's last-received message.

So we need either synchronization between Autocrypt agents on a single account, or there needs to be only one such agent on a given account.

For level 1 and higher, we aim to provide a synchronization mechanism so that all Autocrypt-enabled MUAs connected to a single account are capable of reading encrypted mail.

For simplicitly, level 0 does not require or define synchronization mechanisms, but instead allows an Autocrypt-enable client to "lock" the account so that multiple Autocrypt-enabled clients don't end up sending different keys.

Todo

Describe the tradeoffs and workflow for level-0 agents sharing an account with future level-1 clients, or failure modes (e.g. lockout by an agent you no longer use)

Why do you clamp `Date:` to the current time?

E-mail messages with `Date:` in the future could destroy the ability to update the internal state.

However, since different MUAs view messages at different times, future-dated e-mails could result in state de-synchronization.

Todo

deeper analysis of this state de-sync issue with future-dated e-mails, or alternate, more-stable approaches to dealing with wrong `Date:` headers.

Why do you always encrypt-to-self?

Users expect to be able to read their outbox or Sent Messages folders. Autocrypt should not get in the way of that.

Why did you choose the raw e-mail address for the user ID?

Possibilities for uid we considered:

Option	SC	BC	VO	RvK	SR
no uid				x	x
e-mail	x	x	x	x	
fixed			x	x	x
hash	x		x	x	x

SC: self-claim. This was very important to us for usability reasons. This restricted us to either use the e-mail directly or hashed.

BC: backwards compatibility

VO: valid OpenPGP

RvK: allows revocations using keyservers

SR: Spam resistant/publicly list e-mail addresses

Using a salted hash of the e-mail address for the uid to not list them on keyservers would prevent the privacy issue of public mail addresses but the key should not be uploaded in the first place.

Accidental or malicious uploading of keys with associated e-mail addresses should be prevented by introducing a flag at the keys that says that keyservers shouldn't accept it. See issue #1.

Why RSA2048 and not 25519?

Curve 25519 keys are shorter, cheaper to compute on, and likely to be stronger than RSA 2048 against non-quantum attackers. However, we want level 0 to be implementable in late 2016, and more toolkits support RSA 2048 than 25519. Future versions are likely to encourage 25519 over RSA 2048.

CHAPTER 11

Example User Interface for Autocrypt

CHAPTER 12

Glossary

MUA Mail User Agent. Any program/client/app that handles e-mails for the end user.

MUAA Mail User Agent Account. The data/state a mail user agent holds for a specific account. When synchonizing autocrypt data, we have to synchonize MUAA data (data held by different MUAs for the same account). See *Shared MUAA Messaging Archive*

SMA Shared MUAA Messaging Archive. See *Shared MUAA Messaging Archive*

CHAPTER 13

Future Enhancements to Autocrypt

Please see *Guidance for Implementers of Autocrypt Level 0* for information about Level 0 requirements. Here, we document future improvements, which we hope will be incorporated in Level 1, or possibly some later Level. This is an unordered list. If you have ideas about how to address one of these points, feel free to jump in! (but let's try to stay focused on getting Level 0 stable before we invest too much energy in these next steps)

Expiry

Todo

We need documentation about sensible key expiry policies. Autocrypt-capable clients that choose to have an expiry policy on their secret key material should use message composition as an opportunity to refresh their secret key material or update the expiration dates in their public certificate.

Client sync

Please see *Shared MUAA Messaging Archive*

Todo

We need to specify how to sync internal Autocrypt state between clients. We want to be able to sync the state without sending sync data for every message processed, while we also want all synced peers to have the same internal state as much as possible. We currently believe that syncing updates to `pah` and `changed` should be sufficient, and that peers do not need to sync `last_seen`. This has not been proved in practice.

New Types

Todo

how to deal with multiple types (at least when a new type is specified). When we support types other than *p*, it's possible that users will have multiple keys available, each with a different type. That seems likely to introduce some awkward choices during message composition time, particularly for multi-recipient messages.

X.509 and S/MIME

Todo

Someone is bound to ask for this as a "key type"

Deletable ("forward secure") encrypted mail

Todo

Given the Autocrypt infrastructure for key exchange, there's no reason we couldn't define a mechanism for pairwise, ratcheted, session-key establishment for e-mail.

RSA2048 to Curve 25519

Todo

Document change in preference for keys from RSA 2048 to Curve 25519.

Backups

see *Autocrypt Secret Key Backup*

Todo

We need guidance on how backups might be done safely.

Guidance on masking Key IDs

If any recipients are in *Bcc:* (rather than *To:* or *Cc:*), and the key types used are all OpenPGP (*type=p*), then the agent SHOULD mask the recipient key ID in the generated PKESK packets that correspond to the Bcc'ed recipents. It does not need to mask recipient key IDs of normal recipients.

Masking of Key IDs is done by setting the key ID to all-zeros. See the end of section 5.1 RFC 4880 for more details. Users of GnuPG can use the *–hidden-recipient* argument to indicate a recipient who will be masked.

This is so that the message encryption does not leak much additional metadata beyond what is already found in the headers of the message. It still leaks the number of additional recipients, but the additional work and usability issues involved with fixing that metadata leak suggest that it's better to leave that to a future level.

Encrypted headers

Todo

Document interaction with encrypted headers: if something like memoryhole ever makes it possible to hide normal *To:* and *Cc:* headers, then we need to rethink our approach to handling PKESK leakage further.

Webmail

Todo

How does Autocrypt interact with webmail? Can we describe hooks for webmail and browser extensions that make sense?

Search

Todo

Guidance for implementers on dealing with searching a mailbox that has both cleartext and encrypted messages. (session key caching, etc)

Gossip (or "introduction e-mails")

Todo

Can we specify a sensible practice for passing around keys for other people that we know about?

Or maybe it'd be simpler to define a standard workflow for "introduction e-mails", where the sender tells multiple recipients about the keys she has for all of them.

Out-of-band key verification

Todo

Can we specify a simple, user-friendly way that Autocrypt users can confirm each others' "Autocrypt info" out of band?

If we do specify such a thing, what additional UI/UX would be required?

Heuristics for dealing with "nopreference"

Todo

in Level 0, the Autocrypt recommendations for composing mail to a remote peer with `prefer-encrypted` set to `nopreference` look very much the same as the recommendations for when `prefer-encrypted` is set to `no`. But different heuristics could be applied to the `nopreference` case for MUAs that want to help users be slightly more aggressive about sending encrypted mail.

Documenting reasonable heuristics for MUAs to use in this case would be very helpful.

CHAPTER 14

Autocrypt Secret Key Backup

This is for Autocrypt Level 1 or later...

The MUA generates a strong "backup code" and gets the user to write it down somewhere. Then it serializes its secret key material into a message encrypted by the the backup code. This message is given a custom header and is sent to the account in question:

```
Autocrypt-Secret-Key-Backup: key_backup_data=<encrypted_secret_key>
From: alice@example.net
To: alice@example.net
```

Todo

should the MUA store the message in the SMA, or store it to file or what?

Restore

Todo

Fill in here

Prompting the user for backup code?

Note also that the backup code MUST be strong – it is subject to brute force attacks by anyone who holds a copy.

Backup and Sync

Todo

say something about the relationship between backup and sync

CHAPTER 15

Cleanup needed

RFC 2231 talks about the elements of a MIME header as “parameters” instead of “attributes”. RFC 2045 specifies the same vocab. We should normalize.

Let’s use “cert” where we mean “cert” and “key” where we mean “key”

need a tight document for what is expected of level 0 clients (level0.rst).

user-facing material probably should use “app” – for technical documentation, we need to settle internally on “agent” or “client” or “MUA” or “MUAA”

glossary for technical documentation.

CHAPTER 16

Shared MUAA Messaging Archive

characteristics/requirements of of what SMAs need to provide:

- a SMA can be implemented on top of IMAP commands
- is used to synchronize states between MUAAs. We use "MUAAs" to indicate a particular MUA/Account combination because synchronization happens betweens accounts managed by different MUAs.
- is used to send and receive messages between MUAAs (concurrently), for example pairing requests, initial Autocrypt setup (of first MUAA), updates to received remote Autocrypt encryption keys.
- A MUAA needs to be able to detect if there is any other MUAA
- messages are not (neccesarily) human readable and don't appear in the regular inbox.
- probably: size of SMA should not grow linearly with number of incoming/outgoing mails, for example messages that have been processed by a MUA must be deleted
- there should be a policy/expiry of messages for MUAAs which don't exist/are not alive anymore
- we only require from IMAP servers that they handle first level folders (subfolders are not neccessary)
- there is a header in the messages stored in these folders, indicating that the message is an SMA message.

implementation on top of IMAP, pairing happy path

Let's suppose we have a first MUAA. It doesn't find an `_Autocrypt_SMA` announcement folder so it will do the following:

- create a random new number "1" which we call MUAA-ID.
- create an `_Autocrypt_SMA` "announcements" folder and append some MUAA description message, most notably the MUAA-ID
- create an inbox folder `_Autocrypt_SMA_1` where other MUAAs will be able to send/drop messages.

If now another MUAA is added:

- create a random new number "27" as MUAA-ID.
- discover the `_Autocrypt_SMA` folder exists and read all of its messages, discover that there is an `1` MUAA
- create an inbox folder `_Autocrypt_SMA_27` where other MUAAs will be able to send/drop messages.
- append a new MUAA description message to `_Autocrypt_SMA`
- append a pairing request message to the "1" inbox (`_Autocrypt_SMA_1`).

The MUAA "1" will then:

- discover "27" from the new message in the announcement folder `_Autocrypt_SMA`
- read the pairing request message from its own `_Autocrypt_SMA_1` inbox
- process the pairing request and send a pairing accept message to "27" by appending it to the `_Autocrypt_SMA_27` folder.
- delete the pairing request message from its own `_Autocrypt_SMA_1` folder.

Note: In this happy path example we are not prescribing the precise pairing procedure, merely give an example how bootstrapping into a multi-MUA setting works. It is unclear whether a centrally shared keyring as an IMAP folder is viable (synchronization between MUAs, "merge conflict" between state, deleting message might be a problem, encrypted "broadcast" to all my MUAAs)

Todo

Critically consider how the multiple Autocrypt folders show in user interfaces. It might be better to depend on sub folders.

Todo

Crically consider end-to-end encryption for MUAA messages.

Todo

Consider how to force remove devices through IMAP folder deletion or something.

CHAPTER 17

types of inter-MUAA unicast messages

Difficult to reason about when we don't know what we *really* want to do (cryptographic protocol wise)

ID announcement

pairing messages

- Some authenticated key exchange so later messages between MUAAs can be encrypted
- Shared private key so messages encrypted to the account's public key can be encrypted and outgoing mail can be signed

remote key updates

- notify other MUAAs that you add to or change an entry to your keyring

Index