muacrypt

Release 0.9.2.dev12+g82bb7f0

Jul 24, 2019

Contents

1 Aims and goals 3

2 Documentation, getting started 5

Python Module Index 25

Index 27

muacrypt provides a command line tool and a Python API to help mail agents, both user and server-side, integrate and manage automated e-mail end-to-end encryption with https://autocrypt.org. The project was so far mainly developed by holger krekel (@hpk42) and Azul (@azul) whose work is funded by the European Commission through the NEXTLEAP research project on decentralized messaging. The NEXTLEAP project is concerned with researching and developing secure identity and e2e-encryption protocols and aims to contribute to securing Autocrypt against active attacks.

CHAPTER 1

Aims and goals

- Automatically tested Autocrypt Level 1 compliant API and command line tool, for use by mail user agents (MUAs) and remailers.
- a plugin architecture to integrate other techniques with autocrypt related mail processing. A first example is muacryptcc which implements the decentralized ClaimChain key consistency protocol.
- *integrate muacrypt with mutt* and other MUA setups that call out into commandline tools to automatically achieve e-mail e2e encryption that does not annoy others or yourself (hopefully!)
- integrate `muacrypt` with mailman3 in order to achieve opportunisitically encrypted mailing lists.

CHAPTER 2

Documentation, getting started

2.1 Installation

You need the python package installer "pip". If you don't have it you can install it on Debian systems:

```
sudo apt-get install python-pip
```

And now you can install the muacrypt package:

```
pip install --user muacrypt
```

And then make sure that `~/.local/bin` is contained in your `PATH` variable.

See *getting started with the command line*.

2.1.1 Installation for development

If you plan to work/modify the sources and have a github checkout we recommend to create and activate a python virtualenv and issue **once**:

```
$ cd src
$ virtualenv venv
$ source venv/bin/activate
$ pip install -e .
```

This creates a virtual python environment in the "src/venv" directory and activates it for your shell through the `source venv/bin/activate` command.

Changes you subsequently make to the sources will be available without further installing the autocrypt package again.

2.2 muacrypt command line docs

Note: While the command line tool and its code is automatically tested against gpg, gpg2, python2 and python3, the sub commands are subject to change during the `0.x` releases.

The `muacrypt` command line tool helps to manage Autocrypt information for incoming and outgoing mails for one or more accounts. It follows and implements the Autocrypt spec which defines header interpretation.

Contents

- *muacrypt command line docs*
  - *getting started, playing around*
  - *Using a key from the gpg keyring*
  - *Using separate accounts*
  - *subcommand reference 0.9*
    - *status subcommand*
    - *add-account subcommand*
    - *mod-account subcommand*
    - *del-account subcommand*
    - *process-incoming subcommand*
    - *process-outgoing subcommand*
    - *sendmail subcommand*
    - *test-email subcommand*
    - *make-header subcommand*
    - *export-public-key subcommand*
    - *export-secret-key subcommand*
    - *bot-reply subcommand*
    - *destroy-all subcommand*

2.2.1 getting started, playing around

After *Installation* let's see what sub commands we have:

```
$ muacrypt
Usage: muacrypt [OPTIONS] COMMAND [ARGS]...

  access and manage Autocrypt keys, options, headers.

Options:
  --basedir PATH  directory where muacrypt state is stored
  --version       Show the version and exit.
  -h, --help      Show this message and exit.

Commands:
  status              print account info and status.
  add-account         add named account for set of e-mail...
  mod-account         modify properties of an existing account.
  del-account         delete an account, its keys and all state.
```

(continues on next page)

(continued from previous page)

```
  find-account       print matching account for an e-mail address.
  process-incoming   parse Autocrypt info from stdin message if it...
  scandir-incoming   scan directory for new incoming messages and...
  import-public-key  import public key data as an Autocrypt key.
  peerstate          print current autocrypt state information...
  recommend          print Autocrypt UI recommendation for target...
  process-outgoing   add Autocrypt header for outgoing mail if the...
  sendmail           as process-outgoing but submit to sendmail...
  make-header        print Autocrypt header for an emailadr.
  export-public-key  print public key of own or peer account.
  export-secret-key  print secret key of own account.
  bot-reply          reply to stdin mail as a bot.
  destroy-all        destroy all muacrypt state.
```

For getting started we need to add a new Account:

```
$ muacrypt add-account
account added: 'default'
account: 'default'
  email_regex:     .*
  gpgmode:         own [home: /tmp/home/.config/muacrypt/gpg/default]
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  nopreference
  own-keyhandle:   C40A50563C73AD76
  ^^ uid:           <6403c471d4d440cc83e568e6e4a245b7@random.muacrypt.org>
```

This created a default account which contains a new secret key and a few settings.

Note: If you rather want muacrypt to use your system keyring so that all own and all incoming keys will be stored there, see *syskeyring*.

Let's check out account info again with the `status` subcommand:

```
$ muacrypt status
account-dir: /tmp/home/.config/muacrypt
account: 'default'
  email_regex:     .*
  gpgmode:         own [home: /tmp/home/.config/muacrypt/gpg/default]
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  nopreference
  own-keyhandle:   C40A50563C73AD76
  ^^ uid:           <6403c471d4d440cc83e568e6e4a245b7@random.muacrypt.org>
```

This shows our own keyhandle of our Autocrypt OpenPGP key.

Let's generate a static email Autocrypt header which you could add to your email configuration (substitute `a@example.org` with your email address):

```
$ muacrypt make-header a@example.org
Autocrypt: addr=a@example.org; keydata=
  mQGNBFvwXEwBDADTp/7odJiF7Gm8oKvddUl07QM17qzE8HoMwbYIhFQY9y5Qvi/OOyii1zZz35AH2P
  BaMn0/IrnBknK9JM2klr9qPLKletEDQFs/WrvWekkbFt8CEO4FMJviOY4kCvv5sot462l5lkLh03qs
  r+iURR0jhLJAgb3q8DljPNkIM/1vW3CP5PYyMIBSakzK8J3N3TFfOJnlw6w0sd2M5+DVm8piesWItX
  OxDViNUS6x/0uET2ObrhSw0W7V/j0+/55WMmCxvLz0FBBbDz6nKrPToQtdm+B28azinrsyw0FMt7Q0
  Uw4ogiI9SXygrGZI2IsNWp1JSzeRuVGLZ5dyBCPn+3R2eg//7EK21LGTqpFTSAe0pGOW+N0D6aVI07
  Xb/gpcx7ZFSLycVIsV7dfI+Er3FDVS29zkDQ9SHMTiOxLZYEuA7yF5UXjeZVZVGp+mAdZBZtyAihT5
  0ZI4TRo9PVL93eS9WfnNlAct9L0k5x11zzr4v/IT9NGj/E+DFCUTqq2v2F8AEQEAAbQ3IDw2NDAzYz
  Q3MWQ0ZDQ0MGNjODNlNTY4ZTZlNGEyNDViN0ByYW5kb20ubXVhY3J5cHQub3JnPokBuAQTAQIAIgUC
  W/BcTAIbAwYLCQgHAwIGFQgCCQoLBBYCAwECHgECF4AACgkQxApQVjxzrXanfwv9Fce3thhG+NnOht
  mruC0zVld73FFyUwuY1lDRPK0l8J2mRrIiXi+yB5OVtd1jAmpSz9KYaDTtIjRtAAARQB7/7wbXUTkV
```

(continues on next page)

(continued from previous page)

```
  WDwLn1DRPWyHEeraiCeFvU3fIzQb+KoDr2SfNb+fZC0BVWxBBuesHFFXhBdAY0P49nMuKZq3MvmZxS
  oFTqaVO/9590smS6D3G1bTIW1RhQSo6nPc7VsMRcH4o/6vsx8vl9NJmTaPWASPk45EMAEjKmFAMQiy
  DjFkaduqiDDVsupDEIoSJ2DdexlOi/PmxBBoxIkc31jPzNLd99LGZL26ghCtoEt8ruUeH2ZIY22fS+
  9DEfpH379wcai8U9W6KvcDUO9KtA0cW1OQOQ97P/2uL9KynY8JbrIrTjncgoA0C/0IMR/TF16F4aSa
  ho4OA/LYwvzM2+0cH5vqc40LKT0av2FUGt0lgNcx8vfmJgDBRzJGacJQ6EovrOgxuqDx6pfR0ZE4f0
  jbMtEJT744oqgT8MNIHCV5IT4b1qjeuQGNBFvwXEwBDACzmQrMrP6DAMlUJHFtuD3jLyz+ihZRerwZ
  scKEnnnpYLo5EAUE1SEwVWYub6LtlSZMxeTTAh2VlEvHgh/C8AwYoIw37QYN4zNU8/eh/wTZ76LRiz
  qMuZBX1U6aoe/sKPOzgzjG9V9Pg2RBLpznFFL2VDY8eD9IFClolleaIIHKYyA2ZDM9Pqv4CIswH6W6
  xiNoIh6Sno4wqwBT8paOMVI0g3HcP2d0gFjXO+xBVaILyh/efickcZqpKZeavw3VHKEPOLpRYrE/9L
  VdPUXWFjechHlbHh/cZtIFIMSz05T/O1ydqkAp0HHRyss+VXL8t4NpHumtpdCm7t/Qybgl3XaR14tC
  7bDI2pGq37VzMN3s+wZFTpvBodEiatkpjTYwQykYKM+NF77D9UQpkdyivKllXe0UkePhou8oPIhq1D
  OlEa3xKsq3Hq1WQXgYNLqsA5vK+iAqAPbqBFZDM5j+PWkt4/EwnJaYe4r23BDpLkPxImFIZR5O6up2
  fq/rbgIuHdcAEQEAAYkBnwQYAQIACQUCW/BcTAIbDAAKCRDAEClBWPHOtdnS6DACal+GH6/znjRpswG
  W4NxwMeW0W7s1bEBGva4frFRi12J6Hl95v5gVTgrlPzsCaOO8vYAcLI5fqbu+UsgH40DYjr0YYNIhq
  SrLCKudIW6i69NTj6En48pnieaOFS+HrkV7RSYEh6Vtb//2ESIZ0LXV3El/Zk/MBTFuo5S6ltqBdYG
  +0CKluXCf7ipYS1iBb0OGY4whOt6nrgSUtQwKC7JRe3Hq4tlpn8tu4Q8kMdzhMcVBa2QWDJp6WyFhg
  2iXtqFIPkgaBkQPsxLbrolWEFKXIeJRMyNIV1RB1jJ8WHGextYuOhyK5ysF/ZYG0SmoXiXliwiAIvi
  bs9GW7Vs6tyxljnzo6RmlJoEZvW926bH4j0V1JgDxpcfK0UpyIEU3FhEYsg6eArZi8UnCt6GjyMRRC
  0Mt9DlPAbjxkGfl2NTRhiQXS5SDp7zAJKtLJaRtCWNRfsXlTpd2IwocCxlZi7OsmgQ5G3hC3gQfRf9
  vaqA8jX+X6sHJwL2UnDD2jGgSQq9Y=
```

Getting our own public encryption key in armored format:

```
$ muacrypt export-public-key
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQGNBFvwXEwBDADTp/7odJiF7Gm8oKvddUl07QM17qzE8HoMwbYIhFQY9y5Qvi/O
Oyii1zZz35AH2PBaMn0/IrnBknK9JM2klr9qPLKletEDQFs/WrvWekkbFt8CEO4F
MJviOY4kCvv5sot462l5lkLh03qsr+iURR0jhLJAgb3q8DljPNkIM/1vW3CP5PYy
MIBSakzK8J3N3TFfOJnlw6w0sd2M5+DVm8piesWItXOxDViNUS6x/0uET2ObrhSw
0W7V/j0+/55WMmCxvLz0FBBbDz6nKrPToQtdm+B28azinrsyw0FMt7Q0Uw4ogiI9
SXygrGZI2IsNWp1JSzeRuVGLZ5dyBCPn+3R2eg//7EK21LGTqpFTSAe0pGOW+N0D
6aVI07Xb/gpcx7ZFSLycVIsV7dfI+Er3FDVS29zkDQ9SHMTiOxLZYEuA7yF5UXje
ZVZVGp+mAdZBZtyAihT50ZI4TRo9PVL93eS9WfnNlAct9L0k5x11zzr4v/IT9NGj
/E+DFCUTqq2v2F8AEQEAAbQ3IDw2NDAzYzQ3MWQ0ZDQ0MGNjODNlNTY4ZTZlNGEy
NDViN0ByYW5kb20ubXVhY3J5cHQub3JnPokBuAQTAQIAIgUCW/BcTAIbAwYLCQgH
AwIGFQgCCQoLBBYCAwECHgECF4AACgkQxApQVjxzrXanfwv9Fce3thhG+NnOhtmr
uC0zVld73FFyUwuY1lDRPK0l8J2mRrIiXi+yB5OVtd1jAmpSz9KYaDTtIjRtAAAR
QB7/7wbXUTkVWDwLn1DRPWyHEeraiCeFvU3fIzQb+KoDr2SfNb+fZC0BVWxBBues
HFFXhBdAY0P49nMuKZq3MvmZxSoFTqaVO/9590smS6D3G1bTIW1RhQSo6nPc7VsM
RcH4o/6vsx8vl9NJmTaPWASPk45EMAEjKmFAMQiyDjFkaduqiDDVsupDEIoSJ2Dd
exlOi/PmxBBoxIkc31jPzNLd99LGZL26ghCtoEt8ruUeH2ZIY22fS+9DEfpH379w
cai8U9W6KvcDUO9KtA0cW1OQOQ97P/2uL9KynY8JbrIrTjncgoA0C/0IMR/TF16F
4aSaho4OA/LYwvzM2+0cH5vqc40LKT0av2FUGt0lgNcx8vfmJgDBRzJGacJQ6Eov
rOgxuqDx6pfR0ZE4f0jbMtEJT744oqgT8MNIHCV5IT4b1qjeuQGNBFvwXEwBDACz
mQrMrP6DAMlUJHFtuD3jLyz+ihZRerwZscKEnnnpYLo5EAUE1SEwVWYub6LtlSZM
xeTTAh2VlEvHgh/C8AwYoIw37QYN4zNU8/eh/wTZ76LRizqMuZBX1U6aoe/sKPOz
gzjG9V9Pg2RBLpznFFL2VDY8eD9IFClolleaIIHKYyA2ZDM9Pqv4CIswH6W6xiNo
Ih6Sno4wqwBT8paOMVI0g3HcP2d0gFjXO+xBVaILyh/efickcZqpKZeavw3VHKEP
OLpRYrE/9LVdPUXWFjechHlbHh/cZtIFIMSz05T/O1ydqkAp0HHRyss+VXL8t4Np
HumtpdCm7t/Qybgl3XaR14tC7bDI2pGq37VzMN3s+wZFTpvBodEiatkpjTYwQykY
KM+NF77D9UQpkdyivKllXe0UkePhou8oPIhq1DOlEa3xKsq3Hq1WQXgYNLqsA5vK
+iAqAPbqBFZDM5j+PWkt4/EwnJaYe4r23BDpLkPxImFIZR5O6up2fq/rbgIuHdcA
EQEAAYkBnwQYAQIACQUCW/BcTAIbDAAKCRDAEClBWPHOtdnS6DACal+GH6/znjRps
wGW4NxwMeW0W7s1bEBGva4frFRi12J6Hl95v5gVTgrlPzsCaOO8vYAcLI5fqbu+U
sgH40DYjr0YYNIhqSrLCKudIW6i69NTj6En48pnieaOFS+HrkV7RSYEh6Vtb//2E
SIZ0LXV3El/Zk/MBTFuo5S6ltqBdYG+0CKluXCf7ipYS1iBb0OGY4whOt6nrgSUt
QwKC7JRe3Hq4tlpn8tu4Q8kMdzhMcVBa2QWDJp6WyFhg2iXtqFIPkgaBkQPsxLbr
olWEFKXIeJRMyNIV1RB1jJ8WHGextYuOhyK5ysF/ZYG0SmoXiXliwiAIvibs9GW7
Vs6tyxljnzo6RmlJoEZvW926bH4j0V1JgDxpcfK0UpyIEU3FhEYsg6eArZi8UnCt
```

(continues on next page)

(continued from previous page)

```
6GjyMRRC0Mt9DlPAbjxkGfl2NTRhiQXS5SDp7zAJKtLJaRtCWNRfsXlTpd2IwocC
xlZi7OsmgQ5G3hC3gQfRf9vaqA8jX+X6sHJwL2UnDD2jGgSQq9Y=
=RI6m
-----END PGP PUBLIC KEY BLOCK-----
```

2.2.2 Using a key from the gpg keyring

If you want to use autocrypt with an existing mail setup you can initialize by specifying an existing key in your system gpg or gpg2 key ring. To present a fully self-contained example let's create a standard autocrypt key with gpg:

```
# content of autocrypt_key.spec

Key-Type: RSA
Key-Length: 3072
Key-Usage: sign
Subkey-Type: RSA
Subkey-Length: 3072
Subkey-Usage: encrypt
Name-Email: test@autocrypt.org
Expire-Date: 0
```

Let's run gpg to create this Autocrypt type 1 key:

```
$ gpg --batch --gen-key autocrypt_key.spec
gpg: keyring `/tmp/home/.gnupg/secring.gpg' created
gpg: keyring `/tmp/home/.gnupg/pubring.gpg' created
......+++++
.....+++++
.+++++
.+++++
gpg: /tmp/home/.gnupg/trustdb.gpg: trustdb created
gpg: key 2436BADE marked as ultimately trusted
```

We now have a key generated in the system key ring and can initialize autocrypt using this key. First, for our playing purposes, we delete the current `default` account:

```
$ muacrypt del-account
account deleted: 'default'
account-dir: /tmp/home/.config/muacrypt
no accounts configured
```

and then we add a new default account tied to the key we want to use from the system keyring:

```
$ muacrypt add-account --use-system-keyring --use-key test@autocrypt.org
account added: 'default'
account: 'default'
  email_regex:     .*
  gpgmode:         system
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  nopreference
  own-keyhandle:   DD1E25BE2436BADE
  ^^ uid:           <test@autocrypt.org>
```

Success! We have an initialized autocrypt account with an identity which keeps both our secret and the Autocrypt keys from incoming mails in the system key ring. Note that we created a identity which matches all mail address (`.*`) you might receive mail for or from which you might send mail out. If you rather use aliases or read different accounts from the same folder you may want to look ingo *accounts*.

2.2.3 Using separate accounts

You may want to create separate accounts:

- if you receive mails to alias email addresses in the same folder and want to keep them separate, unlinkable for people who read your mails
- if you read mails from multiple sources in the same folder and want to have Autocrypt help you manage identity separation instead of tweaking your Mail program's config to deal with different Autocrypt accounts.

You can manage accounts in a fine-grained manner. Each account:

- is defined by a name, a regular expression for matching mail addresses and an encryption private/public key pair and prefer-encrypt settings.
- updates Autocrypt peer state from incoming mails if its regex matches the `Delivered-To` address.
- adds Autocrypt headers to outgoing mails if its regex matches the "From" header.

In order to manage an account in a fine grained manner let's start from scratch and delete all `muacrypt` state:

```
$ muacrypt destroy-all --yes
deleting directory: /tmp/home/.config/muacrypt
```

Let's add a new "home" account:

```
$ muacrypt add-account -a home --email-regex '(alice|wonder)@testsuite.autocrypt.
↪org'
account added: 'home'
account: 'home'
  email_regex:     (alice|wonder)@testsuite.autocrypt.org
  gpgmode:         own [home: /tmp/home/.config/muacrypt/gpg/home]
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  nopreference
  own-keyhandle:   51581EF4DD1A3DC1
  ^^ uid:           <151aaad143584c91b29a8b4b3aaf3377@random.muacrypt.org>
```

This creates an decryption/encryption key pair and ties it to the name `home` and a regular expression which matches both `alice@testsuite.autocrypt.org` and `wonder@testsuite.autocrypt.org`.

And now let's create an `office` account:

```
$ muacrypt add-account -a office --email-regex='alice@office.example.org'
account added: 'office'
account: 'office'
  email_regex:     alice@office.example.org
  gpgmode:         own [home: /tmp/home/.config/muacrypt/gpg/office]
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  nopreference
  own-keyhandle:   054E04470D75CE6E
  ^^ uid:           <49fb9c1db86f4b7ea2c23a1fd5f03fbf@random.muacrypt.org>
```

We have now configured two accounts. Let's test if muacrypt matches our `office` address correctly:

```
$ muacrypt find-account alice@office.example.org
office
```

and let's check if muacrypt matches our `home` address as well:

```
$ muacrypt find-account wonder@testsuite.autocrypt.org
home
```

Looks good. Let's modify our `home` account to signal to our peers that we prefer receiving encrypted mails:

```
$ muacrypt mod-account -a home --prefer-encrypt=mutual
account modified: 'home'
account: 'home'
  email_regex:     (alice|wonder)@testsuite.autocrypt.org
  gpgmode:         own [home: /tmp/home/.config/muacrypt/gpg/home]
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  mutual
  own-keyhandle:   51581EF4DD1A3DC1
  ^^ uid:           <151aaad143584c91b29a8b4b3aaf3377@random.muacrypt.org>
```

This new `prefer-encrypt: mutual` setting tells our peers that we prefer to receive encrypted mails. This setting will cause processing of outgoing mails from the home address to add a header indicating that we want to receive encrypted mails if the other side also wants encrypted mails. We can check the setting works with the *make-header* subcommand:

```
$ muacrypt make-header wonder@testsuite.autocrypt.org
Autocrypt: addr=wonder@testsuite.autocrypt.org; prefer-encrypt=mutual; keydata=
  mQGNBFvwXFoBDACulxiA0CAOqxTb0h+hME/hgrXd6jZnA/A8f55F2Qw+q8surWZb/tPqpKepOXI3S+
  V0V8zht/08AGcQNdAG3xR7W87GVyZpxF6vAvQAn96s8jNJ8KiG/UNrIwIJ6rAb9Anj5ouHFaq1Wbn1
  HF/1sqUQnbiw1rztOE2wgmc8ld5aG3WFsDVvf9eefQK0ryIC34Irh5/KsCCRTNPqkPQIVp5uBqJc3y
  KlHCArVoyEQLv3g4D1gNQzXF4VVtOMb6WYqR5dTdpqrfm8Karq+lv5jl4szynj9YUL8P7QHWJNr2Om
  AnVdix9Ju/G0pctlsntlO4k1t4TZM4M4WRg91PeiJN1IhghleCMh8A1VAFkP89uiWzIBucEyZedHY0
  2AnN3Q9hbBNphFzntetQg+Hby3R61cRE2tDAs0i1QzdV7EJEYAphvBcxYx1Dd3X2KxTljTaPUTbicj
  ChcNh8aMv49wVU+TfunGQKFLAxuoBsbKHrVdgpgDHM8txHaMjyPJiVoUyPEAEQEAAbQ3IDwxNTFhYW
  FkMTQzNTg0YzkxYjI5YThiNGIzYWFmMzM3N0ByYW5kb20ubXVhY3J5cHQub3JnPokBuAQTAQIAIgUC
  W/BcWgIbAwYLCQgHAwIGFQgCCQoLBBYCAwECHgECF4AACgkQUVge9N0aPcEF4wv/TbgwfNbhbjNk6i
  ivbbDRAohW3XERUodvOuCwAj2PvxrDI6Rd5h7JpO6YebpciMw9I+K8iKig7OUwSyygbB2zZlNqotQX
  mroYG1tv9i54a6J14SXf6eW5glNOc7RgrDCWneK/yoSo0pLIrFGjpnc1RzRlNTAnaSlyKct1UbWCUp
  R4jj0CAL0xMKT///Q3VcLracsbogYOYI0V5Cf1ih6d4fjfd72zB7a1k2UvQifKut1ZQUPUtf8P2S6V
  Re8RbaZSp7OI5iknZqVjpV3bXPTauI1dLpAd+n2qoAzhbsFHX7hxYfUdHyALkU5r3xaixYA+bZojh6
  rpcLF0MjUTexWubuj32u3GShUOJA+2XetIRPz/Yqp0hTsDsMMUpMAYaIqtSXcCnzcFNK92Xy5ro80j
  ESBcWztz2iBvmLVj83VaEe56oupROtff9Cyh0QDELZ3trHs/s3x0cWSkbG2sQ5disgDjH4pOVR6wrt
  wGeYZDW/uAeM+gUknG29sC9WdknyGLuQGNBFvwXFoBDADZPQFBDP2qyPyoq1HhUK/oTjylDR05Slrf
  JE7VvQkKfejwQk52oolsCX82ixh6fuEA9sHmv++8IL/JpfHZuP0teYHSfgeWC41v9atyEj2ZF1soHn
  uDwxgvy7CGwHx9nw/zYBaigPGSd1Q1gAZz4XAU8tc37GLa7eRwQKkMh6YH5spUZid5qtPlRSJ/SU5s
  x/J2Q0/7kExLB90F2h+j//ataCQNLnLk2ypFbre9rJwXLgLwL8Bcgt0y5oEMgBqFG57iwj9iZberYK
  DX/qx5Xm92A/JszUNJOSDODOrkThILhcizlxFtEnaTK1A3mL5dQlKaoO9kHbUiCSzihTVJgkFrbi+E
  /otooTIDDeqFLx+mRsPEFyRP4r5GiqkUJnduxJsC5W78XtphlEyX71avZQfgK/MFL1i4v3i8geFtBA
  hrlNdqTJH6o/3OuBxt6wfKV6BIWmdHxL+doRkj4TLlDJ8h9rVJILoIxsjNrkMvI28kPw4euXcWA+XJ
  C6lTyCzWNBEAEQEAAYkBnwQYAQIACQUCW/BcWgIbDAAKCRBRWB703Ro9wbXgDACWxBRljSpPH36C1F
  J+K0IGRn1x0ZTw3rAacSKa0hf0goaOl9/pXAwc1LUOW9c9Mwxcv5HYHTi8E3ENQEqakQIsAnsPoRne
  ubgQP8oLGzQJqQ5Y1iC3YRSAYwvhBxLXgWRp429llzEOVw+beXpzrreQ6Wxag6IdOAT/PDQkARnQ8u
  qN59X62WxH2dAM/vQMO8IiXp5FbAZWVtYU6aDLtgrJAFxbifiUEqnaSAcr4otZ3oUEVzmY7oPsAUkr
  cRAM7fEX3sxckeL20K+Q9ddLFPG/Uazwz0ZhNb8o8GcwsmK766QQOx/BL/7R20lD2D9uDdBCF8jQM2
  oSET5uSsd0Fu7JQT5QigAlQ6s6ntTYhkk6N7+gMm2XIsKYN7sJfANv2QVwt2+dT35I/wjCumA6PQTn
  H7MY7QNR6BGRwUOn4uHmEbSzdV+ZS4yLcda2V09uKoy0y/osDLacYhmygzH8Vo/FcGVZsycyMHjOTE
  3q7l0UHBVGfAwx+uT1W8vkOafaFKg=
```

When you pipe a message with a From-address matching Alice's home addresses into the *process-outgoing* subcommand then it will add this header. By using the *sendmail* subcommand (as a substitute for unix's sendmail program) you can cause the resulting mail to be delivered via the `/usr/sbin/sendmail` program.

2.2.4 subcommand reference 0.9

status subcommand

```
Usage: muacrypt status [OPTIONS] [ACCOUNT_NAME]

  print account info and status.
```

(continues on next page)

(continued from previous page)

```
Options:
  -h, --help  Show this message and exit.
```

add-account subcommand

```
Usage: muacrypt add-account [OPTIONS] ACCOUNT_NAME

  add a named account.

  An account requires an account_name which is used to show, modify and
  delete it.

  Of primary importance is the "email_regex" which you typically set to a
  plain email address.   It is used when incoming or outgoing mails need to
  be associated with this account.

  Instead of generating an Autocrypt-compliant key (the default operation)
  you may specify an existing key with --use-key=keyhandle where keyhandle
  may be something for which gpg finds it with 'gpg --list-secret-keys
  keyhandle'. Typically you will then also specify --use-system-keyring to
  make use of your existing keys.  All incoming muacrypt keys will thus be
  statesd in the system key ring instead of an own keyring.

Options:
  --use-key KEYHANDLE   use specified secret key which must be findable
                        through the specified keyhandle (e.g. email, keyid,
                        fingerprint)
  --use-system-keyring  use system keyring for all secret/public keys instead
                        of storing keyring state inside our account directory.
  --gpgbin FILENAME     use specified gpg filename. If it is a simple name it
                        is looked up on demand through the system's PATH.
  --email-regex TEXT    regex for matching all email addresses belonging to
                        this account.
  -h, --help            Show this message and exit.
```

mod-account subcommand

```
Usage: muacrypt mod-account [OPTIONS] ACCOUNT_NAME

  modify properties of an existing account.

  Any specified option replaces the existing one.

Options:
  --use-key KEYHANDLE             use specified secret key which must be
                                  findable through the specified keyhandle
                                  (e.g. email, keyid, fingerprint)
  --gpgbin FILENAME               use specified gpg filename. If it is a
                                  simple name it is looked up on demand
                                  through the system's PATH.
  --email-regex TEXT              regex for matching all email addresses
                                  belonging to this account.
  --prefer-encrypt                ]
                                  modify prefer-encrypt setting, default is to
                                  not change it.
  -h, --help                      Show this message and exit.
```

del-account subcommand

```
Usage: muacrypt del-account [OPTIONS] ACCOUNT_NAME

  delete an account, its keys and all state.

  Make sure you have a backup of your whole account directory first.

Options:
  -h, --help  Show this message and exit.
```

process-incoming subcommand

```
Usage: muacrypt process-incoming [OPTIONS]

  parse Autocrypt headers from stdin mail.

Options:
  -h, --help  Show this message and exit.
```

process-outgoing subcommand

```
Usage: muacrypt process-outgoing [OPTIONS]

  add Autocrypt header for outgoing mail.

  We process mail from stdin by adding an Autocrypt header and send the
  resulting message to stdout. If the mail from stdin contains an Autocrypt
  header we keep it for the outgoing message and do not add one.

Options:
  -h, --help  Show this message and exit.
```

sendmail subcommand

```
Usage: muacrypt sendmail [OPTIONS] [ARGS]...

  as process-outgoing but submit to sendmail binary.

  Processes mail from stdin by adding an Autocrypt header and pipes the
  resulting message to the "sendmail" program. If the mail from stdin
  contains an Autocrypt header we use it for the outgoing message and do not
  add one.

  Note that unknown options and all arguments are passed through to the
  "sendmail" program.

Options:
  -h, --help  Show this message and exit.
```

test-email subcommand

```
Usage: muacrypt test-email [OPTIONS] EMAILADR
```

(continues on next page)

(continued from previous page)

```
  test which account an email belongs to.

  Fail if no account matches.

Options:
  -h, --help  Show this message and exit.
```

make-header subcommand

```
Usage: muacrypt make-header [OPTIONS] EMAILADR

  print Autocrypt header for an emailadr.

Options:
  -h, --help  Show this message and exit.
```

export-public-key subcommand

```
Usage: muacrypt export-public-key [OPTIONS] [KEYHANDLE_OR_EMAIL]

  print public key of own or peer account.

Options:
  -a, --account name  perform lookup through this account
  -h, --help          Show this message and exit.
```

export-secret-key subcommand

```
Usage: muacrypt export-secret-key [OPTIONS]

  print secret key of own account.

Options:
  -a, --account name  perform lookup through this account
  -h, --help          Show this message and exit.
```

bot-reply subcommand

```
Usage: muacrypt bot-reply [OPTIONS]

  reply to stdin mail as a bot.

  This command will generate a reply message and send it to stdout by
  default. The reply message contains an Autocrypt header and details of
  what was found and understood from the incoming mail.

Options:
  --smtp host,port         host and port where the reply should be instead of
                           to stdout.
  --fallback-delivto TEXT  assume delivery to the specified email address if
                           no delivered-to header is found.
  -h, --help               Show this message and exit.
```

destroy-all subcommand

```
Usage: muacrypt destroy-all [OPTIONS]

  destroy all muacrypt state.

  By default this command creates account(s) state in a directory with a
  default "catch-all" account which matches all email addresses and uses
  default settings.  If you want to have more fine-grained control (which
  gpg binary to use, which existing key to use, if to use an existing system
  key ring ...) specify "--no-account".

Options:
  --yes       needs to be specified to actually destroy
  -h, --help  Show this message and exit.
```

2.3 mutt integration with muacrypt

Note: The below mutt/muacrypt integration is a first effort at integration. It may contain errors or rough edges. Please help to refine the integration by making PRs against https://github.com/hpk42/muacrypt and the "doc/mutt.rst" file in particular. Thanks!

`muacrypt` can be used in conjunction with mutt which allows to turn otherwise cleartext mail into encrypted mail. The muacrypt/mutt integration manages PGP keys automatically according to the Autocrypt Level 1 specification. **You don't need to import keys or make decisions about them**.

Apart from *installing muacrypt* you will need to create a muacrypt account and configure the processing of incoming and outgoing mail with your particular mutt/mail setup. The example mutt/muacrypt integration below assumes that you already have a way of synchronizing remote imap folders to a local directory.

We also assume you have a working config for mutt/pgp already and are able to send/receive PGP-encrypted messages. The proposed setup here does not require you to use the PGP- and key-selection menus available from mutt's compose screens. That's exactly what you won't need to care about with the muacrypt/mutt setup. You may also mix your current mutt/PGP usage with Autocrypt usage because muacrypt will leave already encrypted outgoing messages alone.

Contents

- *mutt integration with muacrypt*
  - *Creating an muacrypt account*
  - *Processing outgoing mail / sendmail pipelining*
  - *Controlling encryption through the ENCRYPT header*
  - *Processing incoming mail from maildirs*
  - *Importing existing keys as Autocrypt keys*

2.3.1 Creating an muacrypt account

First, you need to add a new muacrypt Account to keep the Autocrypt state for incoming and outgoing mails and keys. Muacrypt of all accounts is kept in `$HOME/.config/muacrypt` by default.

Because we are working with your existing mutt/pgp integration for being able to decrypt messages it's a good idea to not use muacrypt's default account-creation because this would happen in a separate non-system keyring.

Instead we recommend to use a key from your system keyring when creating the account:

```
muacrypt add-account --use-system-keyring --use-key MY_EMAIL_ADDRESS_OR_KEY_HANDLE
```

You may generate a new dedicated Autocrypt key ("gpg –gen-key") and then reference it for use by mutt/muacrypt instead of re-using an already existing key.

Note: muacrypt does not support secret keys using passphrases. See also Autocrypt's take on it: https://autocrypt.org/level1.html#secret-key-protection-at-rest

---

2.3.2 Processing outgoing mail / sendmail pipelining

The `muacrypt sendmail` command:

- adds Autocrypt headers for outgoing mail from your own address,
- potentially and transparently encrypts outgoing cleartext messages according to the Autocrypt UI recommendation,
- passes on the modified/amended mail to the `sendmail` command.

In your `.muttrc` you need to add something like the following:

```
set sendmail="/path/to/muacrypt sendmail -oem -oi"

# avoid mutt/pgp making decisions about keys now that muacrypt looks
# at each outgoing mail and will itself encrypt if recommended by Autocrypt
set crypt_autoencrypt=no
set crypt_replyencrypt=no
set crypt_replysignencrypted=no
```

The idea here is that that in the composing-mail window you don't work with the mutt/pgp menus at all and let `muacrypt sendmail` do its job of selecting the correct last-seen keys for your recipients. This will also add "Gossip" headers in the encrypted part of outgoing mails so that each of your recipients, if they are using an Autocrypt compliant Mail app, can safely group-reply and maintain encryption.

Todo: currently, `muacrypt sendmail` is not respecting if a mail is a reply to an encrypted mail – Autocrypt recommends to keep replies encrypted in such cases.

2.3.3 Controlling encryption through the ENCRYPT header

Both the `muacrypt sendmail` and `muacrypt process-outgoing` sub commands check for the `ENCRYPT` header in each mail they are processing. The `ENCRYPT` header is only used for internal mutt/muacrypt communication and controls how muacrypt is to treat outgoing messages. The `ENCRYPT` header can have one of three different values:

- `opportunistic` (also the assumed default value if no env-var is present): uses the ui-recommendation of Autocrypt to determine if a mail should be encrypted.
- `yes`: force encrypted mails and fail if encryption is not available for the recipients. Note that forcing encryption can be annoying to **your peer's mail experience** because they might receive mail they can not read in their current situation (webmail/device without secret key).
- `no`: force cleartext even if encryption is recommended.

Note that `muacrypt sendmail` will remove the `ENCRYPT` header after it has processed it and acted accordingly.

It's probably possible to configure mutt keystrokes to set the `ENCRYPT` header during compose but there is no way to show the `ENCRYPT` header in mutt's "compose screen". Therefore the current recommended way for being able to modify/set the ENCRYPT header is:

```
# put into your .muttrc if you want to be able to
# modify the ENCRYPT header for each outgoing mail
my_hdr ENCRYPT: opportunistic
set edit_headers=yes
```

With these settings, when you compose/edit a message you will be able to set the "ENCRYPT" header to one of the above values. However, you don't need to use `edit_headers=yes` – just operating in opportunistic mode without forcing encryption/cleartext will make use of Autocrypt's refined automatic "recommendation" procedures which try to replace cleartext with encrypted mail but only if it is likely that it doesn't get in the way of users.

2.3.4 Processing incoming mail from maildirs

```
$ muacrypt scandir-incoming -h
Usage: muacrypt scandir-incoming [OPTIONS] DIRECTORY

  scan directory for new incoming messages and process Autocrypt and
  Autocrypt-gossip headers from them.

Options:
  -h, --help  Show this message and exit.
```

It is crucial to pipe each new (non-spam) incoming mail to the `muacrypt process-incoming` subcommand, because incoming mails may contain Autocrypt headers both in the cleartext part and the encrypted part of a message.

Unfortunately, mutt's `display_filter` can not be used for calling into `process-incoming` because this hook strips headers that muacrypt needs to see. In the absence of a fitting mutt hook (please suggest one if you know one!) you may use, outside of mutt, a helper command to scan directories for incoming mail:

```
muacrypt scandir-incoming /some/path/to/maildir/
```

All files in the `/some/path/to/maildir` directory will be scanned. If you actually use the Maildir format for your local e-mail copies, it's recommended to only scan mails in the "new" folder:

```
muacrypt scandir-incoming /some/path/to/maildir/new
```

In any case, you need to make sure that `muacrypt scandir-incoming` is invoked every time you have re-synced your local folder from the remote IMAP one. Note that `scandir-incoming` is just a helper which eventually pipes each found mail/file into `muacrypt process-incoming`. If you have other ways of piping new incoming messages through `muacrypt process-incoming` then, by all means, do it and please file a PR against this documentation if it could be of use to other people.

2.3.5 Importing existing keys as Autocrypt keys

If you are already using PGP you might already have keys or get new keys through mail attachments. You can pipe existing keys to muacrypt like this:

```
gpg -a --export SOME_HANDLE_OR_EMAILADR | muacrypt import-public-key
```

Or you can just pipe an attachment from mutt's message-view usually by typing `| muacrypt import-public-key` and you might assign this to a key. Note that the default `muacrypt import-public-key` command will:

- associate all of the email addresses contained in the UIDs with the imported PGP key

- set a prefer-encrypt setting to `mutual` by default.

Please refer to the help for more info on how to change the defaults:

```
$ muacrypt import-public-key -h
Usage: muacrypt import-public-key [OPTIONS]

  import public key data as an Autocrypt key.

  This commands reads from stdin an ascii-armored public PGP key. By default
  all e-mail addresses contained in the UIDs will be associated with the
  key. Use options to change these default behaviours.

Options:
  -a, --account name              use this account name
  --prefer-encrypt [nopreference|mutual]
                                  prefer-encrypt setting for imported key
  --email TEXT                    associate key with this e-mail address
  -h, --help                      Show this message and exit.
```

2.4 Muacrypt API Reference

Note: While the code documented here is automatically tested against gpg, gpg2, python2 and python3, all of the API here could change during `0.x` releases.

muacrypt.account	Account management and processing of incoming / outgoing mails on a per-account basis.
muacrypt.states	All muacrypt states are managed through this module.
muacrypt.chainstore	Storage mechanisms which manage immutable blocks and Chains which consist of hash-linked entries.
muacrypt.mime	Mime message parsing and manipulation functions for Autocrypt usage.
muacrypt.bingpg	BinGPG is a "gpg" or "gpg2" command line wrapper which implements all operations we need for Autocrypt usage.
muacrypt.cmdline	Muacrypt Command line implementation.
muacrypt.bot	Bot command line subcommand to receive and answer with Autocrypt related information for mails to bot@autocrypt.org

2.4.1 account module

Account management and processing of incoming / outgoing mails on a per-account basis.

exception muacrypt.account.**AccountException**
an exception raised during method calls on an AccountManager instance.

class muacrypt.account.**AccountManager**(*dir*, *plugin_manager*)
Manage multiple accounts and route in/out messages to the appropriate account.

__init__(*dir*, *plugin_manager*)
Initialize multi-account configuration.

Parameters

- **dir** (*unicode*) – directory in which muacrypt will states state.

- **plugin_manager** (*pluggy.PluginManager*) – a plugin manager instance with hooks registered

add_account(*account_name=u'default'*, *email_regex=None*, *keyhandle=None*, *gpgbin=u'gpg'*, *gpgmode=u'own'*)
add a named account to this account.

Parameters

- **account_name** – name of this account
- **email_regex** – regular expression which matches all email addresses belonging to this account.
- **keyhandle** – key fingerprint or uid to use for this account.
- **gpgbin** – basename of or full path to gpg binary
- **gpgmode** – "own" (default) keeps all key state inside the account directory under the account. "system" will states keys in the user's system gnupg keyring.

mod_account(*account_name=u'default'*, *email_regex=None*, *keyhandle=None*, *gpgbin=None*, *prefer_encrypt=None*)
modify a named account.

All arguments are optional: if they are not specified the underlying account setting remains unchanged.

Parameters

- **account_name** – name of this account
- **email_regex** – regular expression which matches all email addresses belonging to this account.
- **keyhandle** – key fingerprint or uid to use for this account.
- **gpgbin** – basename of or full path to gpg binary
- **prefer_encrypt** – prefer_encrypt setting for this account.

Returns Account instance

del_account(*account_name*)
fully remove an account.

get_account_from_emailadr(*emailadr*, *raising=False*)
get account for a given email address.

remove()
remove the account directory and re-reset all muacrypt state. You need to add accounts to get working again.

class muacrypt.account.**Account**(*states*, *name*, *plugin_manager*)
An Account manages all Autocrypt settings (both own keys and settings as well as per-peer ones derived from Autocrypt headers).

__init__(*states*, *name*, *plugin_manager*)
shallow initializer. Call create() for initializing this account. exists() tells whether that has happened already.

create(*name*, *email_regex*, *keyhandle*, *gpgbin*, *gpgmode*)
create all settings, keyrings etc for this account.

Parameters

- **name** – name of this account
- **email_regex** – regular expression which matches all email addresses belonging to this account.

- **keyhandle** – key fingerprint or uid to use for this account. If it is None we generate a fresh Autocrypt compliant key.
- **gpgbin** – basename of or full path to gpg binary
- **gpgmode** – "own" keeps all key state inside the account directory under the account. "system" will states keys in the user's system GnuPG keyring.

make_ac_header(*emailadr*)
return Autocrypt header value which uses our own key and the provided emailadr if one of our account matches it.

Parameters **emailadr** (*unicode*) – pure email address which we use as the "addr" attribute in the generated Autocrypt header. An account may generate and send mail from multiple aliases and we advertise the same key across those aliases.

Return type unicode

Returns Autocrypt header value (or empty string)

exists()
return True if the account exists.

export_public_key(*keyhandle=None*)
return armored public key of this account or the one indicated by the key handle.

export_secret_key()
return armored public key for this account.

process_incoming(*msg*, *ignore_existing=False*, *no_decrypt=False*)
process incoming mail message for Autocrypt headers both in the cleartext and encrypted parts which will be decrypted to process Autocrypt-Gossip headers.

Parameters

- **msg** (*email.message.Message*) – instance of a standard email Message.
- **no_decrypt** – if True no encryption (and no gossip keys) will be done.

Return type ProcessIncomingResult or NoneType if message is known already.

import_keydata_as_autocrypt(*addr*, *keydata*, *prefer_encrypt*)
process incoming mail message and states information from any Autocrypt header for the From/Autocrypt peer which created the message.

Parameters

- **addr** (*string or None*) – e-mail address or None if should be extracted from UID of keydata.
- **keydata** (*bytes*) – keydata to be imported

Return type ImportKeyResult

process_gossip_headers(*msg*, *msg_date*, *msg_id*)
process gossip headers from payload mime part of mail message and update state information from any gossip header for the To or Cc recipients of the msg.

Parameters **msg** (*email.message.Message*) – decrypted message returned from decrypt_mime as dec_msg

Return type ProcessIncomingResult

process_outgoing(*msg*)
add Autocrypt header to outgoing message. :type msg: email.message.Message :param msg: outgoing message in mime format. :rtype: ProcessOutgoingResult

encrypt_mime(*msg*, *toaddrs*)
create a new encrypted mime message.

Parameters

- **msg** (*email.message.Message*) – message to be encrypted
- **toaddrs** (*list of e-mail addresses*) – e-mail addresses to encrypt to

Return type *EncryptMimeResult*

decrypt_mime(*msg*)
decrypt an encrypted mime message.

Parameters **msg** (*email.message.Message*) – message to be decrypted

Return type *DecryptMimeResult*

class muacrypt.account.**DecryptMimeResult**(*enc_msg*, *dec_msg*, *keyinfos*)
Result returned from decrypt_mime() with 'enc_msg' (encrypted message) 'dec_msg' (decrypted message) and 'keyinfos' (keys for which the message was encrypted).

__init__(*enc_msg*, *dec_msg*, *keyinfos*)
x.__init__(...) initializes x; see help(type(x)) for signature

class muacrypt.account.**EncryptMimeResult**(*enc_msg*, *keyhandles*)
Result returned from encrypt_mime() with 'msg' (encrypted message) and 'keyhandles' (list of public key handles used for encryption) attributes

__init__(*enc_msg*, *keyhandles*)
x.__init__(...) initializes x; see help(type(x)) for signature

2.4.2 states module

All muacrypt states are managed through this module. We follow the Kappa architecture style (http://milinda.pathirage.org/kappa-architecture.com/) i.e. all state changes are added to append-only chains and they contain immutable entries that may cross-reference other entries (even from other chains). The linking between entries is done using crytographic hashes.

class muacrypt.states.**States**(*dirpath*)
Persisting Muacrypt and per-account settings.

__init__(*dirpath*)

class muacrypt.states.**PeerState**(*chain*)
Synthesized Autocrypt state from parsing messages from a peer.

latest_gossip_entry()
Return latest gossip entry.

__init__(*chain*)
x.__init__(...) initializes x; see help(type(x)) for signature

class muacrypt.states.**OwnState**(*chain*)
Synthesized own state for an account.

__init__(*chain*)
x.__init__(...) initializes x; see help(type(x)) for signature

class muacrypt.states.**OOBState**(*chain*)
Synthesized Out of Band verification state for an account.

__init__(*chain*)
x.__init__(...) initializes x; see help(type(x)) for signature

class muacrypt.states.**AccountManagerState**(*chain*)
Synthesized AccountManagerState.

__init__(*chain*)
x.__init__(...) initializes x; see help(type(x)) for signature

2.4.3 chainstore module

Storage mechanisms which manage immutable blocks and Chains which consist of hash-linked entries.

The HeadTracker keeps track of named "heads" which can be queried through the States class. Both the current BlockService and the HeadTracker use the file system for persistent storage.

class muacrypt.chainstore.**BlockService**(*basedir*)
Filesystem Blockservice for storing and getting immutable blocks for use from Chain instances.

__init__(*basedir*)

class muacrypt.chainstore.**Block**(*cid*, *data*, *bs*)
Basic entry for claim chains. Each Block has the following attributes: - cid: the content address of this block - parent_cid: the parent content address or None - timestamp: when this block was created in seconds since epoch - args: the block-specific payload

__init__(*cid*, *data*, *bs*)

parent
parent block or None.

class muacrypt.chainstore.**HeadTracker**(*path*)
Filesystem implementation for the mutable ID->HEAD mappings

__init__(*path*)

class muacrypt.chainstore.**Chain**(*blockservice*, *headtracker*, *chain_name*)
A Chain maintains an append-only log where each entry in the chain has its own content-based address so that chains can cross-reference entries from the same or other chains. Each entry in a chain carries a timestamp and a parent CID (block hash) and Entry-specific extra data.

__init__(*blockservice*, *headtracker*, *chain_name*)
x.__init__(...) initializes x; see help(type(x)) for signature

2.4.4 mime module

Mime message parsing and manipulation functions for Autocrypt usage.

muacrypt.mime.**parse_email_addr**(*string*)
return the routable email address part from a email-field string.

If the address is of type bytes and not ascii, it is returned in quoted printable encoding.

muacrypt.mime.**parse_ac_headervalue**(*value*)
return a Result object with keydata/addr/prefer_encrypt/extra_attr/error attributes.

If the error attribute is set on the result object then all other attribute values are undefined.

muacrypt.mime.**render_mime_structure**(*msg*, *prefix=u'\u2514'*)
msg should be an email.message.Message object

2.4.5 bingpg module

BinGPG is a "gpg" or "gpg2" command line wrapper which implements all operations we need for Autocrypt usage. It is not meant as a general wrapper outside Autocrypt contexts.

exception muacrypt.bingpg.**InvocationFailure**(*ret*, *cmd*, *out*, *err*, *extrainfo=None*)

__init__(*ret*, *cmd*, *out*, *err*, *extrainfo=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

class muacrypt.bingpg.**BinGPG**(*homedir=None*, *gpgpath=u'gpg'*)
basic wrapper for gpg command line invocations.

exception InvocationFailure(*ret*, *cmd*, *out*, *err*, *extrainfo=None*)

__init__(*ret*, *cmd*, *out*, *err*, *extrainfo=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

__init__(*homedir=None*, *gpgpath=u'gpg'*)

Parameters

- **homedir** (*unicode or None*) – gpg home directory, if None system gpg homedir is used.
- **gpgpath** (*unicode*) – If the path contains path separators and points to an existing file we use it directly. If it contains no path separators, we lookup the path to the binary under the system's PATH. If we can not determine an eventual binary we raise ValueError.

muacrypt.bingpg.**find_executable**(*name*)
return a path object found by looking at the systems underlying PATH specification. If an executable cannot be found, None is returned. copied and adapted from py.path.local.sysfind.

2.4.6 cmdline module

Muacrypt Command line implementation.

2.4.7 bot module

Bot command line subcommand to receive and answer with Autocrypt related information for mails to bot@autocrypt.org

Python Module Index

m

muacrypt.account, 18
muacrypt.bingpg, 22
muacrypt.bot, 23
muacrypt.chainstore, 22
muacrypt.cmdline, 23
muacrypt.mime, 22
muacrypt.states, 21

Index

Symbols

__init__() (*muacrypt.account.Account method*), 19
__init__() (*muacrypt.account.AccountManager method*), 18
__init__() (*muacrypt.account.DecryptMimeResult method*), 21
__init__() (*muacrypt.account.EncryptMimeResult method*), 21
__init__() (*muacrypt.bingpg.BinGPG method*), 23
__init__() (*muacrypt.bingpg.BinGPG.InvocationFailure method*), 23
__init__() (*muacrypt.bingpg.InvocationFailure method*), 22
__init__() (*muacrypt.chainstore.Block method*), 22
__init__() (*muacrypt.chainstore.BlockService method*), 22
__init__() (*muacrypt.chainstore.Chain method*), 22
__init__() (*muacrypt.chainstore.HeadTracker method*), 22
__init__() (*muacrypt.states.AccountManagerState method*), 21
__init__() (*muacrypt.states.OOBState method*), 21
__init__() (*muacrypt.states.OwnState method*), 21
__init__() (*muacrypt.states.PeerState method*), 21
__init__() (*muacrypt.states.States method*), 21

A

Account (*class in muacrypt.account*), 19
AccountException, 18
AccountManager (*class in muacrypt.account*), 18
AccountManagerState (*class in muacrypt.states*), 21
add_account() (*muacrypt.account.AccountManager method*), 19

B

BinGPG (*class in muacrypt.bingpg*), 22
BinGPG.InvocationFailure, 22
Block (*class in muacrypt.chainstore*), 22
BlockService (*class in muacrypt.chainstore*), 22

C

Chain (*class in muacrypt.chainstore*), 22
create() (*muacrypt.account.Account method*), 19

D

decrypt_mime() (*muacrypt.account.Account method*), 21
DecryptMimeResult (*class in muacrypt.account*), 21
del_account() (*muacrypt.account.AccountManager method*), 19

E

encrypt_mime() (*muacrypt.account.Account method*), 20
EncryptMimeResult (*class in muacrypt.account*), 21
exists() (*muacrypt.account.Account method*), 20
export_public_key() (*muacrypt.account.Account method*), 20
export_secret_key() (*muacrypt.account.Account method*), 20

F

find_executable() (*in module muacrypt.bingpg*), 23

G

get_account_from_emailadr() (*muacrypt.account.AccountManager method*), 19

H

HeadTracker (*class in muacrypt.chainstore*), 22

I

import_keydata_as_autocrypt() (*muacrypt.account.Account method*), 20
InvocationFailure, 22

L

latest_gossip_entry() (*muacrypt.states.PeerState method*), 21

M

make_ac_header() (*muacrypt.account.Account method*), 20
mod_account() (*muacrypt.account.AccountManager method*), 19
muacrypt.account (*module*), 18
muacrypt.bingpg (*module*), 22
muacrypt.bot (*module*), 23
muacrypt.chainstore (*module*), 22
muacrypt.cmdline (*module*), 23
muacrypt.mime (*module*), 22
muacrypt.states (*module*), 21

O

OOBState (*class in muacrypt.states*), 21
OwnState (*class in muacrypt.states*), 21

P

parent (*muacrypt.chainstore.Block attribute*), 22
parse_ac_headervalue() (*in module muacrypt.mime*), 22
parse_email_addr() (*in module muacrypt.mime*), 22
PeerState (*class in muacrypt.states*), 21
process_gossip_headers() (*muacrypt.account.Account method*), 20
process_incoming() (*muacrypt.account.Account method*), 20
process_outgoing() (*muacrypt.account.Account method*), 20

R

remove() (*muacrypt.account.AccountManager method*), 19
render_mime_structure() (*in module muacrypt.mime*), 22

S

States (*class in muacrypt.states*), 21