Autocrypt Documentation

Release 0.8.0.dev1

hpk, dkg etc.al

Jan 10, 2018

Contents

1 Installation **3**

2 installation for development **5**

3 Autocrypt command line docs **7**
- 3.1 getting started, playing around . . . 8
- 3.2 Using a key from the gpg keyring . . . 10
- 3.3 Using separate identities . . . 10
- 3.4 subcommand reference 0.8 . . . 12

4 Diagrams **17**
- 4.1 class diagram . . . 17
- 4.2 packages diagram . . . 17

5 Autocrypt Python API Reference **19**
- 5.1 account module . . . 19
- 5.2 cmdline module . . . 19
- 5.3 storage module . . . 19
- 5.4 storage_fs module . . . 19
- 5.5 bot module . . . 20
- 5.6 mime module . . . 20
- 5.7 bingpg module . . . 20

Python Module Index **21**

Note: There is a tentative name change consideration for this project which would result in change of links.

py-autocrypt provides a command line tool and a Python API to help mail agents integrate Autocrypt support and more.

See *Installation* for getting pip-installed with the `autocrypt` package released on the Python Package Index.

Here are some preliminary underlying aims and goals:

- Autocrypt Level 1 compliant functionality for use by mail user agents (MUAs)
- integrate with mailman3 and other server-side mailing software
- provide support for debugging error cases, easy deployment of fixes
- implement out-of-band verification and claimchains variants (see https://nextleap.eu)

The project was so far mainly developed by holger krekel (hpk42) with some participation/contributions from dkg, juga0 and azul. Holger work was and is partially funded by the European Commission through the NEXTLEAP research project on decentralized messaging.

Note that this repository got moved away from the https://github.com/autocrypt umbrella because that is mainly about the Autocrypt specification efforts while MUA/mail related implementations happen through different social arrangements.

CHAPTER 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 autocrypt package:

```
pip install --user autocrypt
```

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

See *getting started with the command line*.

CHAPTER 2

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.

CHAPTER 3

Autocrypt 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 py-autocrypt command line tool helps to manage Autocrypt information for incoming and outgoing mails. It follows and implements the Autocrypt spec and some additional means to make working with it convenient.

Contents

- *Autocrypt command line docs*
  - *getting started, playing around*
  - *Using a key from the gpg keyring*
  - *Using separate identities*
  - *subcommand reference 0.8*
    - *init subcommand*
    - *status subcommand*
    - *add-identity subcommand*
    - *mod-identity subcommand*
    - *del-identity subcommand*
    - *process-incoming subcommand*
    - *process-outgoing subcommand*
    - *sendmail subcommand*
    - *test-email subcommand*
    - *make-header subcommand*
    - *export-public-key subcommand*
    - *export-secret-key subcommand*

3.1 getting started, playing around

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

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

  access and manage Autocrypt keys, options, headers.

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

Commands:
  init               init autocrypt account state.
  status             print account and identity info.
  add-identity       add an identity to this account.
  mod-identity       modify properties of an existing identity.
  del-identity       delete an identity, its keys and all state.
  process-incoming   parse autocrypt headers from stdin mail.
  process-outgoing   add autocrypt header for outgoing mail.
  sendmail           as process-outgoing but submit to sendmail...
  test-email         test which identity an email belongs to.
  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 autocrypt account.
  bot-reply          reply to stdin mail as a bot.
```

For getting started we only need a few commands, first of all we will initialize our Autocrypt account. By default Autocrypt only creates and modifies files and state in its own directory:

```
$ autocrypt init
account directory initialized: /tmp/home/.config/autocrypt
account-dir: /tmp/home/.config/autocrypt

identity: 'default' uuid 64ee038effa649f8a82c22e4d2ec15a4
  email_regex:     .*
  gpgmode:         own [home: /tmp/home/.config/autocrypt/id/default/gpghome]
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  nopreference
  own-keyhandle:   D67E0166618D4146
  ^^ uid:           <64ee038effa649f8a82c22e4d2ec15a4@uuid.autocrypt.org>
  ---- no peers registered -----
```

This created a default identity: a new secret key and a UUID and a few settings. If you rather like autocrypt to use your system keyring so that all incoming keys are available there, see *syskeyring* but this will modify state on your existing keyring.

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

```
$ autocrypt status
account-dir: /tmp/home/.config/autocrypt

identity: 'default' uuid 64ee038effa649f8a82c22e4d2ec15a4
  email_regex:     .*
  gpgmode:         own [home: /tmp/home/.config/autocrypt/id/default/gpghome]
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  nopreference
  own-keyhandle:   D67E0166618D4146
  ^^ uid:           <64ee038effa649f8a82c22e4d2ec15a4@uuid.autocrypt.org>
  ---- no peers registered -----
```

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):

```
$ autocrypt make-header a@example.org
Autocrypt: addr=a@example.org; keydata=
  mQENBFlLz1UBCADM2iM+Nqm8YtHEJYPXBhACycBOalFJAqZzMYUA46xGTop/jBddwgRvNh+ClhQL7H
  xHE+bpfAE0Y1GBfw3PEI/rQGSyY7VhhH6nt7vTHCCYIRP64nfkK/PyRzGGT0AtS40fHc2DZ3kQxG7c
  9krprbmx5fPwudgYzXDY+da7PwNxu9lJyPAjHIfnEsEsxPvTpcChhUs5euifT2sIzJF82UAs0oXqoA
  Ak4G8JF2nZqCILQgkoKlAuEJhw1IjRkOQr19J5UkLKgucNQoOnjJ4HvPdmEt02uqzNXrmUMWl+4Ytb
  XjmaZ3dME6KiH1KbUdTPIhIIVREUnoywslTc+pt5jDEnABEBAAG0NiA8NjRlZTAzOGVmZmE2NDlmOG
  E4MmMyMmU0ZDJlYzE1YTRAdXVpZC5hdXRvY3J5cHQub3JnPokBOAQTAQIAIgUCWUvPVQIbAwYLCQgH
  AwIGFQgCCQoLBBYCAwECHgECF4AACgkQ1n4BZmGNQUZlRQgAr4ZK+0hZ6v65AHu+lw5xa5fIMpSCn6
  anI59VetBur7PbZBIlW5z0jbWW13d+OsS0VW7Uuo07XXzWqc+rpsREpsBa+daWQdi7p/ahLiyd6mhN
  z8WdI+dod/NLmZuDEGllypjveHmbmRreaqIevf5rW6UHhNMReGU91+xHZcbhsqNDYBO/jiUK6EglRt
  zGJJuiJcE3+C/Kqu352OkJQdLDXngkmN2JQsosOmMqIrtPZtVsDHdhljMOOXumbH+G0nJoNNJX25Jv
  iTKdAgaYIcJI5ncEEGVZ6cffN1hPZeM++MvHgnuZ15aWq1cNUXGah27rn/u6pSyKqP0Zq/7RVde+/r
  kBDQRZS89VAQgA5m0ZWf8entimetIOwWj78FZxZldLcZnNKbPiM5sIztTcC2l3my0pfIzDxs9/PIj3
  EE/+u1xPMKWjmU0rh4KRqM1/V7TRbRNOCQhc68OQ3f0yQmeu/B971XHxcslfRm5iV14RFNxbDjyx5O
  IUDSjNy4QBfmMlp1RL81l03Bgv2kalSOPCradEV1eXCE1KSHFu89D6kDjZCZCyd4C+45+T8HdrNfF9
  txy2Lu9quqiiklJDQ3R08ct4WAxMdf5cP/rTdAjRS1ikNR9GwwsHDHnfjVTlz5nknsPl9bTtfIRmRR
  1ijUQaqONRMESYyY9Aq8f0kuhJOdD4y5CccaKBrxti9QARAQABiQEfBBgBAgAJBQJZS89VAhsMAAoJ
  ENZ+AWZhjUFGyfEH/AiFHmaU8XqDJFTkPJX2cfNf8QDPHYio7M++Z15w9y5bp9OU5Amrh8N0Lp+rgv
  262KqED/7FhvMCAljCIF9tk42y/b7jS1hg/qzXfN3wdEbwx1PVqmyZap4PEUXCL97JAjjY+J7D3Yd7
  LQMEN10GdehnWJzuACndx5q2pmkh8u2oHu3Y+XnRUXHm8LMCIrQFx3VTzH0BaWm9kwqVHeAqWpD1tO
  I0kKZx3MVaCcDI7N1JdBwNNqmgBdNhESGUwYd6nHb6tN9c3kGlNfxdNs1v0yXh8B1PwJsTBZPbkC3C
  lx2Sv8FtIICO+e/2pc0PtAtdFARraeeYWgowzzQKZLe/rWc=
```

Getting our own public encryption key in armored format:

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

mQENBFlLz1UBCADM2iM+Nqm8YtHEJYPXBhACycBOalFJAqZzMYUA46xGTop/jBdd
wgRvNh+ClhQL7HxHE+bpfAE0Y1GBfw3PEI/rQGSyY7VhhH6nt7vTHCCYIRP64nfk
K/PyRzGGT0AtS40fHc2DZ3kQxG7c9krprbmx5fPwudgYzXDY+da7PwNxu9lJyPAj
HIfnEsEsxPvTpcChhUs5euifT2sIzJF82UAs0oXqoAAk4G8JF2nZqCILQgkoKlAu
EJhw1IjRkOQr19J5UkLKgucNQoOnjJ4HvPdmEt02uqzNXrmUMWl+4YtbXjmaZ3dM
E6KiH1KbUdTPIhIIVREUnoywslTc+pt5jDEnABEBAAG0NiA8NjRlZTAzOGVmZmE2
NDlmOGE4MmMyMmU0ZDJlYzE1YTRAdXVpZC5hdXRvY3J5cHQub3JnPokBOAQTAQIA
IgUCWUvPVQIbAwYLCQgHAwIGFQgCCQoLBBYCAwECHgECF4AACgkQ1n4BZmGNQUZl
RQgAr4ZK+0hZ6v65AHu+lw5xa5fIMpSCn6anI59VetBur7PbZBIlW5z0jbWW13d+
OsS0VW7Uuo07XXzWqc+rpsREpsBa+daWQdi7p/ahLiyd6mhNz8WdI+dod/NLmZuD
EGllypjveHmbmRreaqIevf5rW6UHhNMReGU91+xHZcbhsqNDYBO/jiUK6EglRtzG
JJuiJcE3+C/Kqu352OkJQdLDXngkmN2JQsosOmMqIrtPZtVsDHdhljMOOXumbH+G
0nJoNNJX25JviTKdAgaYIcJI5ncEEGVZ6cffN1hPZeM++MvHgnuZ15aWq1cNUXGa
h27rn/u6pSyKqP0Zq/7RVde+/rkBDQRZS89VAQgA5m0ZWf8entimetIOwWj78FZx
ZldLcZnNKbPiM5sIztTcC2l3my0pfIzDxs9/PIj3EE/+u1xPMKWjmU0rh4KRqM1/
V7TRbRNOCQhc68OQ3f0yQmeu/B971XHxcslfRm5iV14RFNxbDjyx5OIUDSjNy4QB
fmMlp1RL81l03Bgv2kalSOPCradEV1eXCE1KSHFu89D6kDjZCZCyd4C+45+T8Hdr
NfF9txy2Lu9quqiiklJDQ3R08ct4WAxMdf5cP/rTdAjRS1ikNR9GwwsHDHnfjVTl
z5nknsPl9bTtfIRmRR1ijUQaqONRMESYyY9Aq8f0kuhJOdD4y5CccaKBrxti9QAR
AQABiQEfBBgBAgAJBQJZS89VAhsMAAoJENZ+AWZhjUFGyfEH/AiFHmaU8XqDJFTk
PJX2cfNf8QDPHYio7M++Z15w9y5bp9OU5Amrh8N0Lp+rgv262KqED/7FhvMCAljC
IF9tk42y/b7jS1hg/qzXfN3wdEbwx1PVqmyZap4PEUXCL97JAjjY+J7D3Yd7LQME
N10GdehnWJzuACndx5q2pmkh8u2oHu3Y+XnRUXHm8LMCIrQFx3VTzH0BaWm9kwqV
HeAqWpD1tOI0kKZx3MVaCcDI7N1JdBwNNqmgBdNhESGUwYd6nHb6tN9c3kGlNfxd
Ns1v0yXh8B1PwJsTBZPbkC3Clx2Sv8FtIICO+e/2pc0PtAtdFARraeeYWgowzzQK
ZLe/rWc=
=RDVW
-----END PGP PUBLIC KEY BLOCK-----
```

3.2 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: 2048
Key-Usage: sign
Subkey-Type: RSA
Subkey-Length: 2048
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 4415EEF7 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 recreate the account directory and make sure no default identity is generated:

```
$ autocrypt init --no-identity --replace
deleting account directory: /tmp/home/.config/autocrypt
account directory initialized: /tmp/home/.config/autocrypt
account-dir: /tmp/home/.config/autocrypt
no identities configured
```

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

```
$ autocrypt add-identity default --use-system-keyring --use-key test@autocrypt.org
identity added: 'default'

identity: 'default' uuid 969736e569dc442ab92597fd05e8373c
  email_regex:     .*
  gpgmode:         system
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  nopreference
  own-keyhandle:   F81E1B474415EEF7
  ^^ uid:           <test@autocrypt.org>
  ---- no peers registered -----
```

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 *identities*.

3.3 Using separate identities

You may want to create separate identities with your account:

- 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.

With py-autocrypt you can manage identities in a fine-grained manner. Each identity:

- keeps its autocrypt state in a directory under the account directory.
- is defined by a name, a regular expression for matching mail addresses and an encryption private/public key pair and prefer-encrypt settings.
- stores Autocrypt header information 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 identities in a fine grained manner you need to delete the default identity or to re-initialize your Autocrypt account:

```
$ autocrypt init --no-identity --replace
deleting account directory: /tmp/home/.config/autocrypt
account directory initialized: /tmp/home/.config/autocrypt
account-dir: /tmp/home/.config/autocrypt
no identities configured
```

You can then add an example identity:

```
$ autocrypt add-identity home --email-regex '(alice|wonder)@testsuite.autocrypt.org
↪'
identity added: 'home'

identity: 'home' uuid 1d3bb960f1b347bda83dc3773211a791
  email_regex:     (alice|wonder)@testsuite.autocrypt.org
  gpgmode:         own [home: /tmp/home/.config/autocrypt/id/home/gpghome]
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  nopreference
  own-keyhandle:   23117137B89DE0FB
  ^^ uid:           <1d3bb960f1b347bda83dc3773211a791@uuid.autocrypt.org>
  ---- no peers registered -----
```

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 another identity:

```
$ autocrypt add-identity wonder --email-regex='alice@wunderland.example.org'
identity added: 'wonder'

identity: 'wonder' uuid abebb96743964765af8706f45a4cae76
  email_regex:     alice@wunderland.example.org
  gpgmode:         own [home: /tmp/home/.config/autocrypt/id/wonder/gpghome]
  gpgbin:          gpg [currently resolves to: /usr/bin/gpg]
  prefer-encrypt:  nopreference
  own-keyhandle:   20367F911DD2CA72
  ^^ uid:           <abebb96743964765af8706f45a4cae76@uuid.autocrypt.org>
  ---- no peers registered -----
```

We have now configured our Autocrypt account with two identities. Let's test if Autocrypt matches our `wonder` address correctly:

```
$ autocrypt test-email alice@wunderland.example.org
wonder
```

then one of our `home` ones:

```
$ autocrypt test-email wonder@testsuite.autocrypt.org
home
```

Looks good. Let's modify our `home` identity to signal to its peers that it prefers receiving encrypted mails:

```
$ autocrypt mod-identity home --prefer-encrypt=mutual
Usage: autocrypt mod-identity [OPTIONS] IDENTITY_NAME

Error: Invalid value for "--prefer-encrypt": invalid choice: yes. (choose from␣
↪nopreference, mutual)
```

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:

```
$ autocrypt make-header wonder@testsuite.autocrypt.org
Autocrypt: addr=wonder@testsuite.autocrypt.org; keydata=
  mQENBFlLz1kBCADd4K43W/x/im2sASRoURw9Pxa2uz+aiebGQnuz6+fOJMmcJl2MRIsQVh6vKpPuOh
  qE9JLGqgxbgv9oaC97RgY00JCeabXHAsE0OY9AXsyaGmur1BLp0kV4IE+sqHZWtqudT/F+7FDxdkMN
  +Zsv4Ek5w6iLBkNleD3XJB58pFJNelhOrUaJEgVcxwvblx05tXerC2nIgjSclirND8EfXGV499E+lF
  jcmmDMt+OvLSg5U/dB4u9k3seThlWItT+zqHjl+m1sSK0rKq7p+lfMkqFNIAlGVcU/TG+QbgfhfoLC
  r28M1+M36ydmDZMHmvf1wunKd02rF8deVc5Nl8PxBDCpABEBAAG0NiA8MWQzYmI5NjBmMWIzNDdiZG
  E4M2RjMzc3MzIxMWE3OTFAdXVpZC5hdXRvY3J5cHQub3JnPokBOAQTAQIAIgUCWUvPWQIbAwYLCQgH
  AwIGFQgCCQoLBBYCAwECHgECF4AACgkQIxFxN7id4PuIUAf/aJEJQcBTnpwYkT57NjM74LUTGEmE8E
  lvclRpj+b/+SBbECMMyLbUgklk3do8K2mmWdei12tJtsBSXvFy1ZB0JWZ5PXSLcy8CAAJGtp2GShvC
  3z4x7WDfgMX/HJgMfexUIL8Q+kUwPuRVo5CU+Po0l3E/huSpmRoGEJMeZGAtI07F9OxffYBcEsKI4q
  fzug3ID9wDZQoX2zNZB/9998BhZI1d0e2/acnux7aedDsMxu3sAj/kVd8WRifPxW2//L+oqhP6/s+H
  8vo1jHIOUFyFMfNLzeU1+puyKmRMNM13tFjC9gCJ/pskieI1DMtMVA4LNdNF9fRGbEg1lSrg6zaZ5r
  kBDQRZS89ZAQgAtmeWmxdYh8O1kkgp/wJL/GGKKPHMxJnuXO+rFecW4j/S3u1dmU84Z5Iz1o31Py9b
  aOM2xv3ylbqTnLINNqf+2BjXbVRyTf3vuXIOxwbsMRcZmI+tOdc+CDIjceq5Hr7jWCTT9diBiMSCmE
  fSLyWykAZpBINbmgmXTk53wRsn6WoiU6CGGs1fOn5gcKQWgzHDPX7764XEOM9ShJgGMYLYfESyrJbK
  /c3f49mh2TN4u+6l27KHxCWt/bC+FcADYeS+b/YvVz0vNlmgx+0SCXDq0V9VA4tWPDhewDTK/E5itU
  iH2UUJg0WYZRT3yWwleQuKu+ctQnrOEYIUOeWwkEzicwARAQABiQEfBBgBAgAJBQJZS89ZAhsMAAoJ
  ECMRcTe4neD7e8IIAJQh5oNB0CkYnMn6uSBp2ePF9hId8SIIflSX6vHCbLt394VByb3VNeQgfZ3oRk
  1ZzPHAPnEw7OoV5momM5JoR8lset3vt5LJamUcNCuQsjgZwD5pfhrJO5qgfARaKskTtAX8/2oKDznI
  HDFFtAhAd45cegE4UL5fkNQzQat0z84jAiSk+F6cCdGpFPaLApMoQTOLmnGfk9KSIORu/7fsvw3m9f
  76m1/UKCwJRPGaIwIOgTaXfhzUM/pyXFp/JoHJchKaLBbbJimfwNvzUj3YkUm4O57qnHF07tXnojSN
  rCGPzrHYIP092Sm2w1V54VV3q0aVpF/P6UCna7SNWDzxiEg=
```

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

3.4 subcommand reference 0.8

3.4.1 init subcommand

init:

Usage: autocrypt init [OPTIONS]

init autocrypt account state.

By default this command creates account state in a directory with a default "catch-all" identity 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-identity".

Options:

--replace	delete autocrypt account directory before attempting init
--no-identity	initializing without creating a default identity
-h, --help	Show this message and exit.

3.4.2 status subcommand

status:

Usage: autocrypt status [OPTIONS]

print account and identity info.

Options:

-h, --help	Show this message and exit.

3.4.3 add-identity subcommand

add-identity:

Usage: autocrypt add-identity [OPTIONS] IDENTITY_NAME

add an identity to this account.

An identity requires an identity_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 identity.

Instead of generating a 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 autocrypt keys will thus be stored 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 identity 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 identity.
-h, --help	Show this message and exit.

3.4.4 mod-identity subcommand

mod-identity:

Usage: autocrypt mod-identity [OPTIONS] IDENTITY_NAME

modify properties of an existing identity.

An identity requires an identity_name.

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 identity.

--prefer-encrypt modify prefer-encrypt setting, default is to not change it.

-h, --help Show this message and exit.

3.4.5 del-identity subcommand

del-identity:

Usage: autocrypt del-identity [OPTIONS] IDENTITY_NAME

delete an identity, 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.

3.4.6 process-incoming subcommand

process-incoming:

Usage: autocrypt process-incoming [OPTIONS]

parse autocrypt headers from stdin mail.

Options:

-h, --help Show this message and exit.

3.4.7 process-outgoing subcommand

process-outgoing:

Usage: autocrypt 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.

3.4.8 sendmail subcommand

sendmail:

Usage: autocrypt 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.

3.4.9 test-email subcommand

test-email:

Usage: autocrypt test-email [OPTIONS] EMAILADR

test which identity an email belongs to.

Fail if no identity matches.

Options:

-h, --help Show this message and exit.

3.4.10 make-header subcommand

make-header:

Usage: autocrypt make-header [OPTIONS] EMAILADR

print autocrypt header for an emailadr.

Options:

-h, --help Show this message and exit.

3.4.11 export-public-key subcommand

export-public-key:

Usage: autocrypt export-public-key [OPTIONS] [KEYHANDLE_OR_EMAIL]

print public key of own or peer account.

Options:

--id identity perform lookup through this identity

-h, --help Show this message and exit.

3.4.12 export-secret-key subcommand

export-secret-key:

Usage: autocrypt export-secret-key [OPTIONS]

print secret key of own autocrypt account.

Options:

--id identity perform lookup through this identity

-h, --help Show this message and exit.

CHAPTER 4

Diagrams

4.1 class diagram

(updated: 2017-11-07)

4.2 packages diagram

(updated: 2017-11-07)

CHAPTER 5

Autocrypt Python API Reference

Note: While the code documented here is automatically tested against gpg, gpg2, python2 and python3, all of the API here is subject to change during `0.x` releases. This doesn't mean that everything will actually change.

`autocrypt.account`	
`autocrypt.storage`	
`autocrypt.storage_fs`	HeadTracker and BlockService filesystem implementation.
`autocrypt.bingpg`	BinGPG is a "gpg" or "gpg2" command line wrapper which implements all operations we need for Autocrypt usage.
`autocrypt.cmdline`	
`autocrypt.bot`	
`autocrypt.mime`	mime message parsing and manipulation functions for Autocrypt usage.

5.1 account module

5.2 cmdline module

5.3 storage module

5.4 storage_fs module

HeadTracker and BlockService filesystem implementation.

5.5 bot module

5.6 mime module

mime message parsing and manipulation functions for Autocrypt usage.

autocrypt.mime.**parse_ac_headervalue**(*value*)
return a autocrypt attribute dictionary parsed from the specified autocrypt header value. Unspecified default values for prefer-encrypt and the key type are filled in.

autocrypt.mime.**parse_email_addr**(*string*)
return a (prefix, emailadr) tuple.

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

autocrypt.mime.**verify_ac_dict**(*ac_dict*)
return a list of errors from checking the autocrypt attribute dict. if the returned list is empty no errors were found.

5.7 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.

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

__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.

autocrypt.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.

Python Module Index

a

autocrypt.bingpg, 20
autocrypt.mime, 20
autocrypt.storage_fs, 19

Index

Symbols

__init__() (autocrypt.bingpg.BinGPG method), 20

A

autocrypt.bingpg (module), 20
autocrypt.mime (module), 20
autocrypt.storage_fs (module), 19

B

BinGPG (class in autocrypt.bingpg), 20

F

find_executable() (in module autocrypt.bingpg), 20

P

parse_ac_headervalue() (in module autocrypt.mime), 20
parse_email_addr() (in module autocrypt.mime), 20

R

render_mime_structure() (in module autocrypt.mime), 20

V

verify_ac_dict() (in module autocrypt.mime), 20