Autocrypt Documentation

Release 0.7.0

hpk, dkg etc.al

Sep 27, 2017

Contents

1 Autocrypt command line docs **3**
- 1.1 getting started, playing around . . . 4
- 1.2 Using a key from the gpg keyring . . . 6
- 1.3 Using separate identities . . . 6
- 1.4 subcommand reference 0.7 . . . 8

2 Autocrypt Python API Reference **13**
- 2.1 account module . . . 13
- 2.2 bot module . . . 16
- 2.3 mime module . . . 16
- 2.4 bingpg module . . . 16
- 2.5 pgpycrypto module . . . 16
- 2.6 claimchain module . . . 16

3 Installation **19**

4 installation for development **21**

Python Module Index **23**

Note: The py-autocrypt tool is as much in development as the spec itself. Until we have a 1.0 release everything is subject to change.

CHAPTER 1

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 0.7*
    - *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*

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-----
```

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*.

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.

subcommand reference 0.7

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.

status subcommand

status:

Usage: autocrypt status [OPTIONS]

print account and identity info.

Options:

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

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.

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.

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.

process-incoming subcommand

process-incoming:

Usage: autocrypt process-incoming [OPTIONS]

parse autocrypt headers from stdin mail.

Options:

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

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.

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.

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.

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.

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.

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 2

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	Contains Account class which offers all autocrypt related access and manipulation methods.
autocrypt.bot	
autocrypt.mime	mime message parsing and manipulation functions for Autocrypt usage.
autocrypt.bingpg	BinGPG is a "gpg" or "gpg2" command line wrapper which implements all operations we need for Autocrypt usage.
autocrypt.pgpycrypto	

account module

Contains Account class which offers all autocrypt related access and manipulation methods. It also contains some internal helpers which help to persist config and peer state.

exception autocrypt.account.**AccountException**
an exception raised during method calls on an Account instance.

class autocrypt.account.**Account**(*dir*)
Autocrypt Account class which allows to manipulate autocrypt configuration and state for use from mail processing agents. Autocrypt uses a standalone GPG managed keyring and persists its config to a default app-config location.

You can init an account and then use it to generate Autocrypt headers and process incoming mails to discover and memorize a peer's Autocrypt headers.

__init__(*dir*)
Initialize the account configuration and internally used gpggrapper.

Parameters

- **dir** (*unicode*) – directory in which autocrypt will store all state including a gpg-managed keyring.
- **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.

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

Parameters

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

mod_identity(*id_name=u'default'*, *email_regex=None*, *keyhandle=None*, *gpgbin=None*, *prefer_encrypt=None*)
modify a named identity.

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

Parameters

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

Returns Identity instance

del_identity(*id_name*)
fully remove an identity.

get_identity_from_emailadr(*emailadr_list*, *raising=False*)
get identity for a given email address list.

remove()
remove the account directory and reset this account configuration to empty. You need to add identities to reinitialize.

make_header(*emailadr*, *headername=u'Autocrypt: '*)
return an Autocrypt header line which uses our own key and the provided emailadr if this account is managing the emailadr.

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.
- **headername** (*unicode*) – the prefix we use for the header, defaults to "Autocrypt". By specifying an empty string you just get the header value.

Return type unicode

Returns autocrypt header with prefix and value (or empty string)

process_incoming(*msg*, *delivto=None*)
process incoming mail message and store information from any Autocrypt header for the From/Autocrypt peer which created the message.

Parameters **msg** (*email.message.Message*) – instance of a standard email Message.

Return type *PeerInfo*

process_outgoing(*msg*)
process outgoing mail message and add Autocrypt header if it doesn't already exist.

Parameters **msg** (*email.message.Message*) – instance of a standard email Message.

Return type *PeerInfo*

class autocrypt.account.**Identity**(*dir*)
An Identity manages all Autocrypt settings and keys for a peer and stores it in a directory. Call create() for initializing settings.

__init__(*dir*)

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

Parameters

- **name** – name of this identity
- **email_regex** – regular expression which matches all email addresses belonging to this identity.
- **keyhandle** – key fingerprint or uid to use for this identity. 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 identity directory under the account. "system" will store keys in the user's system GnuPG keyring.

get_peerinfo(*emailadr*)
get peerinfo object for a given email address.

Parameters **emailadr** (*unicode*) – pure email address without any prefixes or real names.

Return type *PeerInfo* or None

exists()
return True if the identity 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.

class autocrypt.account.**PeerInfo**(*identity*, *d*)
Read-Only info coming from the Parsed Autocrypt header from an incoming Mail from a peer. In addition to the public Autocrypt attributes (`addr`, `keydata`, `type`, ...) we process also py-autocrypt internal `*date` and `*keyhandle` attributes.

__init__(*identity*, *d*)

class autocrypt.account.**IdentityInfo**(*name*, *email_regex*, *prefer_encrypt*, *keyhandle*, *peers*, *uuid*)
Read only information about an Identity in an account.

__init__(*name*, *email_regex*, *prefer_encrypt*, *keyhandle*, *peers*, *uuid*)

bot module

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.

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'*)

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.

pgpycrypto module

Note: The "pgpy" backend is tested but not used not used yet because pgpy==0.4.1 are not sufficiently substituting gpg functionality yet.

claimchain module

Note: The claimchain module is not required for, or part of Autocrypt Level 1. It is a prototype and experimental effort which azul and hpk are playing with to allow for helping users protect against MITM attacks or, conversely, to make it more costly for providers or network-level attackers who want to subvert communications. This is part of their involvement on the NEXTLEAP EU project.

Eventually claimchains could be integrated as a Plugin but this requires an according pluginization of py-autocrypt which is better to do after the prototyping stabilizes.

Basic ClaimChain implementation.

Each claimchain is associated with an externally provided identifier which is, however, not part of the claimchain itself. We presume each claimchain instance relates either to an own account or to a remote peer (email address), both of which are represented through a unique identifier.

The storage infrastructure for claimchains works by creating and accessing immutable blocks through the BlockService. Each block serves as a ClaimChain entry which is conceptually an append-only log. The current head (last item of the log) associated with each identifier is obtained and managed through a HeadTracker instance. Both the BlockService and the HeadTracker use the file system for persistent storage. If we could use IPFS libs (see below todo) and a subset of their infrastructure we might also use a distributed global BlockService without much coding change. To protect blocks from public reading we can add symmetric encryption and transfer the according secret in-band as well.

Each claimchain instance starts with a "genesis" entry which contains an Autocrypt public key. When receiving a claimchain from someone it should be framed within a signature with this genesis key. One way to achieve this is to send ClaimChains only within encrypted&signed messages.

Another claimchain entry type is "oob_verification" which expresses successful out-of-band verification of claimchain heads and key material between two users.

todo/to-consider:

- properly implement oob verification
- look into using ipfs's modules/concepts for serializing and creating "content ids", i.e. self-describing hash addresses to blocks
- add crypto signing of each entry? For in-band transmission of ClaimChains we can probably just sign the whole chain instead of the single entries.

CHAPTER 3

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.

CHAPTER 4

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.

Python Module Index

a

autocrypt.account, 13
autocrypt.bingpg, 16
autocrypt.claimchain, 16
autocrypt.mime, 16

Index

Symbols

__init__() (autocrypt.account.Account method), 13
__init__() (autocrypt.account.Identity method), 15
__init__() (autocrypt.account.IdentityInfo method), 15
__init__() (autocrypt.account.PeerInfo method), 15
__init__() (autocrypt.bingpg.BinGPG method), 16

A

Account (class in autocrypt.account), 13
AccountException, 13
add_identity() (autocrypt.account.Account method), 14
autocrypt.account (module), 13
autocrypt.bingpg (module), 16
autocrypt.claimchain (module), 16
autocrypt.mime (module), 16

B

BinGPG (class in autocrypt.bingpg), 16

C

create() (autocrypt.account.Identity method), 15

D

del_identity() (autocrypt.account.Account method), 14

E

exists() (autocrypt.account.Identity method), 15
export_public_key() (autocrypt.account.Identity method), 15
export_secret_key() (autocrypt.account.Identity method), 15

F

find_executable() (in module autocrypt.bingpg), 16

G

get_identity_from_emailadr() (autocrypt.account.Account method), 14
get_peerinfo() (autocrypt.account.Identity method), 15

I

Identity (class in autocrypt.account), 15
IdentityInfo (class in autocrypt.account), 15

M

make_header() (autocrypt.account.Account method), 14
mod_identity() (autocrypt.account.Account method), 14

P

parse_ac_headervalue() (in module autocrypt.mime), 16
parse_email_addr() (in module autocrypt.mime), 16
PeerInfo (class in autocrypt.account), 15
process_incoming() (autocrypt.account.Account method), 15
process_outgoing() (autocrypt.account.Account method), 15

R

remove() (autocrypt.account.Account method), 14
render_mime_structure() (in module autocrypt.mime), 16

V

verify_ac_dict() (in module autocrypt.mime), 16