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.
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](#).
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:

```bash
$ 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):

```bash
$ autocrypt make-header a@example.org
Autocrypt: addr=a@example.org; keydata=
mQENBFlLz1UBCADM2iM+Nqm8YtHEJYPXbAhAyEcBoALfJAgzZMYUA46xGTop/jBddwvRvNh+ClhQL7hHBEbpaEOY1GBf3W3FjRI/0qg5SY7Vhnh6nt7VTHCCY1RP64nFk/KFyrZGOTUAT5OSlEFc2DZ5m2QXg7c9kprrbm5fewgZydXa7dy7a7PwnKnxu9yjPYAhfnEsExvPtpChhUs5eiulitfSizJF8w0a0XqoAPqk4G8JF2n2zQ1lGqkKlAeuHw1j1rrkQrJ95uk1KgucQNoQonj4HyVpmEtr70uzqXrnmUWl+4YtbjXj3Z3meDkHKnudtpf1h1VREUnowysItc+ptj5DENaEBEBAAAG0NIAn8jRL2atCOWvznME2ND1mOG4mMumUYm00ZJLy1yU3TAVwC5hdxKrvy3J5cHqb3j3nPoCBAQTAqvAg1UCwUCVwpVqIvbAhvLcQhAxWigFOGQcCQLOBYCAwEChGEFCA4ACqkgkn4BzmGnuKZ1IQqgr42k+0r6z6v65Ahu+iw5xa5f1MpmCn6aiv595VetB7pZB11sWz0jwBN1w3d+0sOS0W7uU074XzzWg+rpsRePsBa+daWqi7p/ahLiyd6mhnzn8wQd+402mU2EdG1lyypewBnhrmreaqeyev59W6UHhWNMrEGU91+kxZcbhsqNDBYD/jiUK6EglRtjG0jUjicE3+C/kQ/)5z05KQjD6DXgknmn2JQoaosOmqItrFZtvSdWdh1jMNOXumbHb8G0jNoNUNX5JVj/1TkXaYaYcJ5ncCEEGVZ6c6cF1nh1PzE+M+NVhgpu15awq1cNUXGah27rz/u6y8yKqP0q7/Rvd+e/RBDQRZS9V9AqgAsm0ZzwNtmetiem10wq7F8xZLdCnZKpbPjM5aeIztC253my0pf1d9S/PI3JEE+/u1xPMKWpKmr0u4HkRq9m7V7TrbRNCOQch68B0q3f0yQumeq/B971XHcscsflm5i1V4RFBDbjxy50IUD5n5y4Qbfsm1P1RBL11M2V2kaiPSCrdEvlECE1KSHPu896dkJjZCZcyd4c4s++T8HdrrngF9txy2Lqunj0DjQ380ct4AwAmdfc5f/rdAjrSl1KrnRgwGswdHwnfHvb1t5lznkspn19BtffmRmRRrjUjQaogONRMESyY99aqf0kohJODd45CcEtkBrxtJ7QARQAABIqEBfBqAgAQBqJZS89AhvMAAOJENZ+AWHjIfgFyEH/AfLhM8uaXqJfDkFJX2CfNFQDHPHyo7M++/z39y59b90u5Amhr8N01P+rqv262qKnEd7hF/MACJ1CFT9k4y2/b3j1Shq/qxv3wEebwx1PVyq1z2acpe3UPEUXCL97AJyj+J7D3yd7LQMEN10gdehnJwuAcndx5mqpmhxu2huoh3y+XnXURHM8LMCIrQFX3V7Z80aw9KnvJwAaqmgWp1OCOII0KZkXnMcAcDI7N1JdbBNnGWdWNy6dHbN6t93c3GlIinfxsnlsv0yXhB1PWjsTSBZPBkC31cx2s8ftIICO+e/2pC0PttAdtFARraeeYWgOwzzQKZLe/rWc=
```

Getting our own public encryption key in armored format:

```bash
$ autocrypt export-public-key
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1
mQENBFlLz1UBCADM2iM+Nqm8YtHEJYPXbAhAyEcBoALfJAgzZMYUA46xGTop/jBddwvRvNh+ClhQL7hHBEbpaEOY1GBf3W3FjRI/0qg5SY7Vhnh6nt7VTHCCY1RP64nFk/KFyrZGOTUAT5OSlEFc2DZ5m2QXg7c9kprrbm5fewgZydXa7dy7a7PwnKnxu9yjPYAhfnEsExvPtpChhUs5eiulitfSizJF8w0a0XqoAPqk4G8JF2n2zQ1lGqkKlAeuHw1j1rrkQrJ95uk1KgucQNoQonj4HyVpmEtr70uzqXrnmUWl+4YtbjXj3Z3meDkHKnudtpf1h1VREUnowysItc+ptj5DENaEBEBAAAG0NIAn8jRL2atCOWvznME2ND1mOG4mMumUYm00ZJLy1yU3TAVwC5hdxKrvy3J5cHqb3j3nPoCBAQTAqvAg1UCwUCVwpVqIvbAhvLcQhAxWigFOGQcCQLOBYCAwEChGEFCA4ACqkgkn4BzmGnuKZ1IQqgr42k+0r6z6v65Ahu+iw5xa5f1MpmCn6aiv595VetB7pZB11sWz0jwBN1w3d+0sOS0W7uU074XzzWg+rpsRePsBa+daWqi7p/ahLiyd6mhnzn8wQd+402mU2EdG1lyypewBnhrmreaqeyev59W6UHhWNMrEGU91+kxZcbhsqNDBYD/jiUK6EglRtjG0jUjicE3+C/kQ/)5z05KQjD6DXgknmn2JQoaosOmqItrFZtvSdWdh1jMNOXumbHb8G0jNoNUNX5JVj/1TkXaYaYcJ5ncCEEGVZ6c6cF1nh1PzE+M+NVhgpu15awq1cNUXGah27rz/u6y8yKqP0q7/Rvd+e/RBDQRZS9V9AqgAsm0ZzwNtmetiem10wq7F8xZLdCnZKpbPjM5aeIztC253my0pf1d9S/PI3JEE+/u1xPMKWpKmr0u4HkRq9m7V7TrbRNCOQch68B0q3f0yQumeq/B971XHcscsflm5i1V4RFBDbjxy50IUD5n5y4Qbfsm1P1RBL11M2V2kaiPSCrdEvlECE1KSHPu896dkJjZCZcyd4c4s++T8HdrrngF9txy2Lqunj0DjQ380ct4AwAmdfc5f/rdAjrSl1KrnRgwGswdHwnfHvb1t5lznkspn19BtffmRmRRrjUjQaogONRMESyY99aqf0kohJODd45CcEtkBrxtJ7QARQAABIqEBfBqAgAQBqJZS89AhvMAAOJENZ+AWHjIfgFyEH/AfLhM8uaXqJfDkFJX2CfNFQDHPHyo7M++/z39y59b90u5Amhr8N01P+rqv262qKnEd7hF/MACJ1CFT9k4y2/b3j1Shq/qxv3wEebwx1PVyq1z2acpe3UPEUXCL97AJyj+J7D3yd7LQMEN10gdehnJwuAcndx5mqpmhxu2huoh3y+XnXURHM8LMCIrQFX3V7Z80aw9KnvJwAaqmgWp1OCOII0KZkXnMcAcDI7N1JdbBNnGWdWNy6dHbN6t93c3GlIinfxsnlsv0yXhB1PWjsTSBZPBkC31cx2s8ftIICO+e/2pC0PttAdtFARraeeYWgOwzzQKZLe/rWc=
```

3.1. getting started, playing around
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' uid 969736e569dc442ab92597fd05e8373c
e-mail regex: .*
gpgmode: system
gpgbin: gpg [currently resolves to:/usr/bin/gpg]
prefer-encrypt: nopreference
own-keyhandle: F81E1B474415EEF7
^ ^ uuid: <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 into 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
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
prefer-encrypt: nopreference
own-keyhandle: 20367F911D2CA72
^ ^ 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
```

3.3. Using separate identities
then one of our home ones:

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

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+fOJMmcJl2MRIsQVh6vKpPuOhqE9jLGqqxbvg9oaC979y00JCeabXHaseE0Y9AxsyaGmurlB1pl0kV4IE=sqH2Zwqtd/F+7FDxdkMN+2sv4EK5w61BbKnelD3XJ58BpFJNei1orUaJEgVcxwvblx0t5xrC2nI9jScll1rNDBEFV499E+1FjcmmdMDT+OlvL5q/0Bu49K3sethlWIt+zqHj1+ms5Kr0Kq7p+1fmkqFNIA1GVCUT/G+QbghfoLC
```

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+fOJMmcJl2MRIsQVh6vKpPuOhqE9jLGqqxbvg9oaC979y00JCeabXHaseE0Y9AxsyaGmurlB1pl0kV4IE=sqH2Zwqtd/F+7FDxdkMN+2sv4EK5w61BbKnelD3XJ58BpFJNei1orUaJEgVcxwvblx0t5xrC2nI9jScll1rNDBEFV499E+1FjcmmdMDT+OlvL5q/0Bu49K3sethlWIt+zqHj1+ms5Kr0Kq7p+1fmkqFNIA1GVCUT/G+QbghfoLC
```

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”.

12 Chapter 3. Autocrypt command line docs
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:

- `--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:

- `--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:

- `--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

- `--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

- `--help`
  Show this message and exit.
4.1 class diagram

(updated: 2017-11-07)

4.2 packages diagram

(updated: 2017-11-07)
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.

<table>
<thead>
<tr>
<th>Autocrypt Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>autocrypt.account</code></td>
<td></td>
</tr>
<tr>
<td><code>autocrypt.storage</code></td>
<td></td>
</tr>
<tr>
<td><code>autocrypt.storage_fs</code></td>
<td>HeadTracker and BlockService filesystem implementation.</td>
</tr>
<tr>
<td><code>autocrypt.bingpg</code></td>
<td>BinGPG is a “gpg” or “gpg2” command line wrapper which implements all operations we need for Autocrypt usage.</td>
</tr>
<tr>
<td><code>autocrypt.cmdline</code></td>
<td></td>
</tr>
<tr>
<td><code>autocrypt.bot</code></td>
<td></td>
</tr>
<tr>
<td><code>autocrypt.mime</code></td>
<td>mime message parsing and manipulation functions for Autocrypt usage.</td>
</tr>
</tbody>
</table>

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.

```python
import autocrypt

class autocrypt.mime:
    def parse_ac_headervalue(self, value):
        return parse_ac_headervalue(value)

    def parse_email_addr(self, string):
        return parse_email_addr(string)

    def render_mime_structure(self, msg, prefix="\u2514"):
        return render_mime_structure(msg, prefix="\u2514")

    def verify_ac_dict(self, ac_dict):
        return verify_ac_dict(ac_dict)
```

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.

```python
import autocrypt

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

    def find_executable(self, name):
        return find_executable(name)
```

20 Chapter 5. Autocrypt Python API Reference
Python Module Index

a

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