**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.
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:

```bash
$ 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:

```bash
$ autocrypt init
account directory initialized: /tmp/home/.config/autocrypt
account-dir: /tmp/home/.config/autocrypt
identity: 'default' uuid 64ee038e6fa649f8e82c22e4d2ec15a4
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: <64ee038e6fa649f8e82c22e4d2ec15a4@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:

```bash
$ autocrypt status
account-dir: /tmp/home/.config/autocrypt
identity: 'default' uuid 64ee038e6fa649f8e82c22e4d2ec15a4
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: <64ee038e6fa649f8e82c22e4d2ec15a4@uuid.autocrypt.org>
---- no peers registered -----
```
Autocrypt Documentation, Release 0.7.0

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+Nqm8YtHEYJPxhAcycBoaIFJAgz2MYUA46xGTop/jBdwdqVnRh+Ci1hQ7LxhE+bpBAe01GYBFw3FPEI/rqG5Y7Vhhh6nc7VVTHCY1RPF46nfk/FyPyRzGTOA540FcH2DZ3kXG7c9Kprrbmx5FwpFwpyZnDZyDA+yda7FmWhnX91YjPaJHIfnEsEsPvTpcChhUs5euiFtZs1zjF82u3oXxqA Ak4G6F2n2zg4LqgkKlaEuWn1ljrkQo19J5ukLkucNOoNj4HvPdmEt02uqzNXnUMW1+4Ytb XjzA3dM6EkiNhKbdTPf7I1VREUnowys1ct+5t5JDeNAEBBAA0NA9YJL1TAzOOGvznME2ND1MG E4MmMym0Uz02JZ1y11TlYcA3pLcXhKXvRY3J5cRbNub3Jn3k0BaQTAjA1qUcUCWVdPQIVAvBAlwLQCh AwIFQGfCQClCBWCAwCZEGyFC44ACGkGk2N4BzmGNQ2U10gqRf42k+0h26655AhUr1+1w5x5aF1Mps6n 6anl15v95v0rU17p7BzBl115z0jqBNWd13oVs0S0W7Uu07XZzXqG+rpzRpsEpRsBa+daQwDi7p/ahLiyd6bNh z8Wdi+0d/LNmZeDGlylph/vehmNhrRaqevleF5W6UHnNREguQ91+xWzcbshqNDB0/jIuU6Eg1Rt zGOJU1jCJ3c+3/C/qq350kQjDLDxGnm2JQOsOmvMqWhFzTsVdsDWHdj3MOOXumhB+G0JJoNXXJ5Jv I7kDqAgay1ClCJ5nCEEGVZ6cF1NhPEmm+MvHgWtu15aQw1cNUXGAh27zn/u6P8yXKpQ0q/7Rvdve+/x KBDQZRSS9VAgA5m0Zm7bemt10Ww7pXZ2dLcZnNKBpJH5seizCTzc231my0fPzix9/PI13 EE+/u1xPMKwJmu0U7r4rQML/V7TVbRrNQNCQhc860Q89fOyQmeu/B971HXcsIn5m2i1WRFxvDj5yXO IUDS5Jny4QBfm1m1PRL1103B2ka1SOPCradEvlXExEKSHPu89D6KdjZZCy4c4+450T8HrdRfF9 ttxyL2u9jqi4c3kR038ct4WAxmf5dvc/S+rDArJS11knR9GwHwDhHfJVTj52nspr19TbfFmRM 11juQoaQNMRESSEYY99f80kUOd45C5cCaBKrxti1Q9ARAQABQfE8FggAWJQJSZ989HavmMAJoJ ENZ+AWHhJFpgyFh/AFLhMau8XqDfTKPjX2cF8PDPHYi7M+115w9Y5b90UAmrh8N0L1pv+qV 262KqEd7F/7vMCAJ1CFnt4y2/bj731ihq/gzqF3WeXEBwX1PVmgYzap4PEUXCL97AJyj+J7D3y7 ztQfM10GdehVjWzuAcdn5q2pmKhu2U0hU3Y+xNruXHm8LMC1rQF3vXtZ0Hawbn6k9wVqehAwpDqK c1O 1lKZX3MvAcdB71NjDBwNMNgDwYuYd6bH6tN93cG1nFxdnSlv0yX8h1BPwhJSTsBPBc3C 1x2Sv8FIICO+c/2pcOOpTAtdFARaeeyWagwzzQKZLe/rVe/
```

Getting our own public encryption key in armored format:

```
$ autocrypt export-public-key
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1
mQENBFlLz1UBCADM2lM+Nqm8YtHEYJPxhAcycBoaIFJAgz2MYUA46xGTop/jBdwdqVnRh+Ci1hQ7LxhE+bpBAe01GYBFw3FPEI/rqG5Y7Vhhh6nc7VVTHCY1RPF46nfk/FyPyRzGTOA540FcH2DZ3kXG7c9Kprrbmx5FwpFwpyZnDZyDA+yda7FmWhnX91YjPaJHIfnEsEsPvTpcChhUs5euiFtZs1zjF82u3oXxqA Ak4G6F2n2zg4LqgkKlaEuWn1ljrkQo19J5ukLkucNOoNj4HvPdmEt02uqzNXnUMW1+4Ytb XjzA3dM6EkiNhKbdTPf7I1VREUnowys1ct+5t5JDeNAEBBAA0NA9YJL1TAzOOGvznME2ND1MG E4MmMym0Uz02JZ1y11TlYcA3pLcXhKXvRY3J5cRbNub3Jn3k0BaQTAjA1qUcUCWVdPQIVAvBAlwLQCh AwIFQGfCQClCBWCAwCZEGyFC44ACGkGk2N4BzmGNQ2U10gqRf42k+0h26655AhUr1+1w5x5aF1Mps6n 6anl15v95v0rU17p7BzBl115z0jqBNWd13oVs0S0W7Uu07XZzXqG+rpzRpsEpRsBa+daQwDi7p/ahLiyd6bNh z8Wdi+0d/LNmZeDGlylph/vehmNhrRaqevleF5W6UHnNReGuQ91+xWzcbshqNDB0/jIuU6Eg1Rt zGOJU1jCJ3c+3/C/qq350kQjDLDxGnm2JQOsOmvMqWhFzTsVdsDWHdj3MOOXumhB+G0JJoNXXJ5Jv I7kDqAgay1ClCJ5nCEEGVZ6cF1NhPEmm+MvHgWtu15aQw1cNUXGAh27zn/u6P8yXKpQ0q/7Rvdve+/x KBDQZRSS9VAgA5m0Zm7bemt10Ww7pXZ2dLcZnNKBpJH5seizCTzc231my0fPzix9/PI13 EE+/u1xPMKwJmu0U7r4rQML/V7TVbRrNQNCQhc860Q89fOyQmeu/B971HXcsIn5m2i1WRFxvDj5yXO IUDS5Jny4QBfm1m1PRL1103B2ka1SOPCradEvlXExEKSHPu89D6KdjZZCy4c4+450T8HrdRfF9 ttxyL2u9jqi4c3kR038ct4WAxmf5dvc/S+rDArJS11knR9GwHwDhHfJVTj52nspr19TbfFmRM 11juQoaQNMRESSEYY99f80kUOd45C5cCaBKrxti1Q9ARAQABQfE8FggAWJQJSZ989HavmMAJoJ ENZ+AWHhJFpgyFh/AFLhMau8XqDfTKPjX2cF8PDPHYi7M+115w9Y5b90UAmrh8N0L1pv+qV 262KqEd7F/7vMCAJ1CFnt4y2/bj731ihq/gzqF3WeXEBwX1PVmgYzap4PEUXCL97AJyj+J7D3y7 ztQfM10GdehVjWzuAcdn5q2pmKhu2U0hU3Y+xNruXHm8LMC1rQF3vXtZ0Hawbn6k9wVqehAwpDqK c1O 1lKZX3MvAcdB71NjDBwNMNgDwYuYd6bH6tN93cG1nFxdnSlv0yX8h1BPwhJSTsBPBc3C 1x2Sv8FIICO+c/2pcOOpTAtdFARaeeyWagwzzQKZLe/rVe/
```

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

Getting our own public encryption key in armored format:

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

1.1. getting started, playing around
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:

```plaintext
# 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:

```shell
$ gpg --batch --gen-key autocrypt_key.spec
```

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:

```shell
$ autocrypt init --no-identity --replace
```

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

```shell
$ autocrypt add-identity default --use-system-keyring --use-key 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 into 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
```

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

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
```

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".

8 Chapter 1. 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.

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.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autocrypt.account</td>
<td>Contains Account class which offers all autocrypt related access and manipulation methods.</td>
</tr>
<tr>
<td>autocrypt.bot</td>
<td></td>
</tr>
<tr>
<td>autocrypt.mime</td>
<td>mime message parsing and manipulation functions for Autocrypt usage.</td>
</tr>
<tr>
<td>autocrypt.bingpg</td>
<td>BinGPG is a “gpg” or “gpg2” command line wrapper which implements all operations we need for Autocrypt usage.</td>
</tr>
<tr>
<td>autocrypt.pgpycrypto</td>
<td></td>
</tr>
</tbody>
</table>

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.

```python
def add_identity(id_name='default', email_regex='.*', keyhandle=None, gpgbin='gpg', gpgmode='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.

```python
def mod_identity(id_name='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

```python
def del_identity(id_name):
    fully remove an identity.
```

```python
def get_identity_from_emailadr(emailadr_list, raising=False):
    get identity for a given email address list.
```

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

```python
def make_header(emailadr, headername='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)

```python
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:

Return type: PeerInfo

```python
process_outgoing(msg)
```

process outgoing mail message and add Autocrypt header if it doesn’t already exist.

Parameters:

Return type: PeerInfo

```python
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.

```python
__init__(dir)
```

```python
class autocrypt.account.IdentityInfo(name, email_regex, prefer_encrypt, keyhandle, peers, uuid)
```

Read only information about an Identity in an account.

2.1. account module

15

---

**Return type**  unicode

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

```python
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**

**Return type**  PeerInfo

```python
process_outgoing(msg)
```

process outgoing mail message and add Autocrypt header if it doesn’t already exist.

**Parameters**

**Return type**  PeerInfo

```python
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.

```python
__init__(dir)
```

```python
class autocrypt.account.IdentityInfo(name, email_regex, prefer_encrypt, keyhandle, peers, uuid)
```

Read only information about an Identity in an account.

2.1. account module 15
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=\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 substitut-
ing 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.
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.
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.
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
delete_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_header_value() (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

25