trousseau Documentation

Release 0.3.0

Oleiade

March 08, 2016

Contents

Trousseau is a **gpg** encrypted key-value store designed to be a *simple*, *safe* and *trustworthy* place for your data. It stores data in a single multi-recipients encrypted file and can supports both local and remote storage sources (S3 and ssh so far) import/export.

Create a *trousseau* store, specify which *gpg* recipients are allowed to open and modify it, add some key-value pairs to it, export it to S3 for example, and re-import it on another device. As simple as that.

Whether you're a devops, a paranoid guy living in a bunker, or the random user who seeks a simple way to store it's critical data in secured manner. *Trousseau* can do something for you.

CHAPTER 1

Setting things up

1.1 Installation

1.1.1 Debian and ubuntu

A binary debian repository provides *trousseau* packages for *i386*, *x86_64* and *arm* architectures, so you can easily install it. Just add the repository to your sources.list:

```
$ echo "deb http://dl.bintray.com/oleiade/deb /" | sudo tee /etc/apt/sources.list.d/trousseau.list
```

And you're ready to go:

```
$ sudo apt-get update && sudo apt-get install trousseau
```

1.1.2 OSX

Homebrew

If you're using homebrew just proceed to installation using the provided formula:

```
$ brew install trousseau.rb
```

Et voila!

Macports

Coming soon (Don't be shy, if you feel like you could do it, just send pull request ;))

Build it

1. First, make sure you have a Go language compiler **>= 1.1.2** (*mandatory*) and git installed.
2. Make sure you have the following go system dependencies in your `$PATH`: *bzr, svn, hg, git*
3. Then, just build and copy the `./bin/trousseau` executable to a system *PATH* location

```
make
sudo cp ./bin/trousseau /usr/local/bin/trousseau
```

1.2 Prerequisities

1.2.1 Gpg passphrase

Every decryption operations will require your *gpg* primary key passphrase.

As of today, **trousseau** is able to handle your passphrase through multiple ways:

- system's keyring manager
- gpg-agent daemon
- system environment
- `--passphrase` global option

Keyring manager

Supported system keyring manager are osx keychain access and linux gnome secret-service and gnome-keychain (more might be added in the future on demand). To use the keyring manager you will need to set up the `TROUSSEAU_KEYRING_SERVICE` environment variable to the name of they keyring manager key holding the trousseau main gpg key passphrase.

```
$ export TROUSSEAU_KEYRING_SERVICE=my_keyring_key
$ trousseau get abc
```

Gpg agent

Another authentication method supported is gpg-agent. In order to use it make sure you've started the gpg-agent daemon and exported the `GPG_AGENT_INFO` variable, trousseau will do the rest.

```
$ export GPG_AGENT_INFO=path_to_the_gpg_agent_info_file
$ export TROUSSEAU_MASTER_GPG_ID=myid@mymail.com
$ trousseau get abc
```

Environment variable

Alternatively, you can pass your primary key passphrase as `TROUSSEAU_PASSPHRASE` environment variable:

```
$ export TROUSSEAU_PASSPHRASE=mysupperdupperpassphrase
$ trousseau get abc
```

Passphrase global option

Ultimately, you can pass you gpg passphrase through the command line global option:

```
$ trousseau --passhphrase mysupperdupperpassphrase get abc
```

1.2.2 Environment

Trousseau behavior can be controlled through the system environment:

- *TROUSSEAU_STORE* : if you want to have multiple trousseau data store, set this environment variable to the path of the one you want to use. Default is `$HOME/.trousseau`

CHAPTER 2

Usage

2.1 Store creation

First use of **trousseau** requires the data store to be created. A **trousseau** data store is built and maintained for a list of *gpg* recipients who will be the only ones able to decrypt and manipulate it (so don't forget to include yourself ;))

2.1.1 API

- **create** [RECIPIENTS ...] : creates the trousseau encrypted datastore for provided recipients and stores it in `$HOME/.trousseau`
- **meta** : Outputs the store metadata.
- **add-recipient** RECIPIENT : Adds a recipient to the store. The recipient will be able to open and modify the store.
- **remove-recipient** RECIPIENT : Removes a recipient from the store. The recipient will not be able to open or modify the store.

2.1.2 First steps with the data store

```
$ trousseau create 4B7D890,28EA78B  # create a trousseau for two gpg recipients
trousseau created at $HOME/.trousseau
```

Trousseau data store consists in single gpg encrypted file residing in your `$HOME` directory. Check by yourself.

```
$ cat ~/.trousseau
-----BEGIN PGP MESSAGE-----
wcBMA5i2a4x3jHQgAQgAGKAZd5UFauGBMkFz7wi4v4aNTGGpDS81drrevo/Tntdz
rr+PR/GjUlKZxhvG18mr+FuTV6q2DOK3Z0nROs57PLK9Q3ye40Su/Af1vj+LaN4i
AAMK9YVpjKaxz+pciUm8nBDkRxp3CLZ9eA2B+1JBy5HgziHY+7KC/dvaubRv0M0J
qzYvshIYU0urVQt7oO4WYVQbJ1N0OXV3oAzW4bBBs/p6b8KSUlmvHUr+9r4V1KvU
ynpHbp1T2HVPC9uqLgJ+PRjlQ2QsxjezkBntOFMaeMZjq2m2glw90aIGDAPjkMKy
42qQbmdrT3+houqeKUrLcVFNOxevVEZLf8N3Qgo/H9LgAeSroddqYkJzOmknxDzP
MDk+4TaY4Ljge+G7j+CB4iBsIjrgSefl/4ZU30dJ/DHyL5i3lCCGXXAo2eqfJg2w
FZgh+qc8Mbjlz2iMdnC+b8rRwhMTgD1Tyd8vbR1ArPfQh3ThdePwrdyE86CYQZOA
MIBfKgTUpWiAtEhM23melF8H3oznrIKt1ZtDsxJEuBCZ86XlC9TF27XFWbnl7rfK
jF2kqP3DuuBA5d23HprbN6LjDSJeKbXDvc5LetBI7O5y954n3tMWCB9y4EjkpVAx
EWnovjEnnW89uXHaFOBQ4naH4kjg1OHEquCf4Nvgl+S5Pfi875yAKqxxK/+e8GGo
4q8UZC7ho/cA
```

```
=t2zr
-----END PGP MESSAGE-----
```

2.2 Keys manipulation

Once your trousseau has been created, you're now able to read, write, list, delete its data. Here's how the fun part goes.

2.2.1 API

- **get** KEY [–file]: Outputs the stored KEY-value pair, whether on *stdout* or in pointed `--file` option path.
- **set** KEY [VALUE | –file] : Sets the provided key-value pair in store using provided value or extracting it from path pointed by `--file` option.
- **del** KEY : Deletes provided key from the store
- **keys** : Lists the stored keys
- **show** : Lists the stored key-value pairs

2.2.2 You've got the keys

```
# Right now the store is empty
$ trousseau show

# Let's add some data into it
$ trousseau set abc 123
$ trousseau set "easy as" "do re mi"

# set action supports a --file flag to use the content
# of a file as value
$ trousseau set myuser.ssh.public_key --file ~/.ssh/id_rsa.pub

# Now let's make sure data has been added
$ trousseau keys
abc
easy as
myuser.ssh.public_key --file ~/.ssh/id_rsa.pub

$ trousseau get abc
123

$ trousseau show
abc: 123
easy as: do re mi
myuser.ssh.public_key: ssh-rsa 1289eu102ij30192u3e0912e
...

# Whenever you want to export a key value to a file, just use
# the get command --file option
$ trousseau get myuser.ssh.public_key --file /home/myuser/id_rsa.pub

# Now if you don't need a key anymore, just drop it.
$ trousseau del abc  # Now the song lacks something doesn't it?
```

2.3 Remote storage import/export

Trousseau was built with data remote storage in mind. Therefore it provides *push* and *pull* actions to export and import the trousseau data store to remote destinations. As of today S3 and SSH storages are available (more are to come). Moreover,

2.3.1 API

- **push** : Pushes the trousseau data store to remote storage
- **pull** : Pulls the trousseau data store from remote storage

2.3.2 DSN

In order to make your life easier trousseau allows you to select your export and import sources using a *DSN*.

- **protocol**: The remote service target type. Can be one of: *s3* or *scp*
- **identifier**: The login/key/whatever to authenticate **trousseau** to the remote service. Provide your *aws_access_key* if you're targeting *s3*, or your remote login if you're targeting *scp*.
- **secret**: The secret to authenticate **trousseau** to the remote service. Provide your *aws_secret_key* if you're targeting *s3*, or your remote password if you're targeting *scp*.
- **host**: Your bucket name is you're targeting *s3*. The host to login to using *scp* otherwise.
- **port**: The *aws_region* if you're targeting *s3*. The port to login to using *scp* otherwise.
- **path**: The remote path to push to or retrieve from the trousseau file on a `push` or `pull` action.

2.3.3 S3 Example

```
# Considering a non empty trousseau data store
$ trousseau show
abc: 123
easy as: do re mi

# And then you're ready to push
$ trousseau push s3://aws_access_key:aws_secret_key@bucket:region/remote_file_path


# Now that data store is pushed to S3, let's remove the
# local data store and pull it once again to ensure it worked
$ rm ~/.trousseau
$ trousseau show
Trousseau unconfigured: no data store

$ trousseau pull s3://aws_access_key:aws_secret_key@bucket:region/remote_file_path
$ trousseau show
abc: 123
easy as: do re mi
```

2.3.4 Scp example

```
# We start with a non-empty trousseau data store
$ trousseau show
abc: 123
easy as: do re mi

# To push it using scp we need to provide it a couple of
# basic options
$ trousseau push scp://user:password@host:port/remote_file_path


# Now that data store has been pushed to the remote storage
# using scp, let's remove the local data store and pull it
# once again to ensure it worked
$ rm ~/.trousseau
$ trousseau show
Trousseau unconfigured: no data store

$ trousseau pull scp://user:password@host:port/remote_file_path
$ trousseau show
abc: 123
easy as: do re mi
```

2.4 Local imports and exports

2.4.1 API

- **import** FILENAME: will import a trousseau data store from the local filesystem. The operation **erases** the current trousseau store content.
- **export** FILENAME: will export the current trousseau data store as *FILENAME* on the local fs.

2.4.2 Real world example

```
$ trousseau export testtrousseau.asc  # Fine we've exported our current data store into a single fil
$ mail -f testtrousseau.asc cousin@machin.com  # Let's pretend we've sent it by mail

# Now cousin machin is now able to import the data store
$ trousseau import testtrousseau.asc
$ trousseau show
cousin_machin:isagreatbuddy
adams_family:rests in peace, for sure
```

2.5 Metadata

Trousseau keeps track and exposes all sort of metadata about your store that you can access through the `meta` command.

```
$ trousseau meta
CreatedAt: 2013-08-12 08:00:20.457477714 +0200 CEST
LastModifiedAt: 2013-08-12 08:00:20.457586991 +0200 CEST
```

```
Recipients: [4B7D890,28EA78B]
TrousseauVersion: 0.1.0c
```

Once again, if you're intersted in how the meta data are stored, go check yourself by decrypting the store content using one of your recipients private key.

```
$ cat ~/.trousseau | gpg -d -r 4B7D890 --textmode
You need a passphrase to unlock the secret key for
user: "My Gpg User <MyGpg@mail.com>"
2048-bit RSA key, ID 4B7D890, created 2013-05-21 (main key ID 4B7D890)

gpg: encrypted with 2048-bit RSA key, ID 4B7D890, created 2013-05-21
"My Gpg User <MyGpg@mail.com>"
{"_meta":{"created_at":"2013-08-12 08:00:20.457477714 +0200 CEST","last_modified_at":"2013-08-12 08:0
```

2.5.1 Adding and removing recipients

Okay, so you've created a trousseau data store with two recipients allowed to manipulate it. Now suppose you'd like to add another recipient to be able to open and update the trousseau store; or to remove one. `add-recipient` and `remove-recipient` commands can help you with that.

```
$ trousseau add-recipient 75FE3AB
$ trousseau add-recipient 869FA4A
$ trousseau meta
CreatedAt: 2013-08-12 08:00:20.457477714 +0200 CEST
LastModifiedAt: 2013-08-12 08:00:20.457586991 +0200 CEST
Recipients: [4B7D890, 75FE3AB, 869FA4A]
TrousseauVersion: 0.1.0c

$ trousseau remove-recipient 75FE3AB
$ trousseau meta
CreatedAt: 2013-08-12 08:00:20.457477714 +0200 CEST
LastModifiedAt: 2013-08-12 08:00:20.457586991 +0200 CEST
Recipients: [4B7D890, 869FA4A]
TrousseauVersion: 0.1.0c
```

CHAPTER 3

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/oleiade/trousseau/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

3.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

3.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

3.1.4 Write Documentation

Trousseau could always use more documentation, whether as part of the official Trousseau docs, in docstrings, or even on the web in blog posts, articles, and such.

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/oleiade/trousseau/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.2 Get Started!

Ready to contribute? Here's how to set up *trousseau* for local development.

1. Fork the *trousseau* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/trousseau.git
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, check that your changes pass tests, and is properly formatted:

```
$ make tests
$ make vet
$ make fmt
```

5. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request to the *develop* branch through the GitHub website.

3.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the README.rst.
3. The pull request should work for go >= 1.2. Check https://travis-ci.org/oleiade/trousseau/pull_requests and make sure that the tests pass for all supported go versions.