



build unknown

## What

*Trousseau* is an encrypted key-value store designed to be a *simple, safe and trustworthy* place for your data.

It stores data in a **single encrypted file**. It supports both **asymmetric encryption** using OpenPGP, and **symmetric encryption** using AES256. It can be easily synced across devices using Dropbox, OneDrive... It can be exported and imported to/from multiple remote storages using integrated S3, ssh, and gist support. If used with OpenPGP encryption, it is able to restrict access to the data store to a set of recipients.

Create a *trousseau* data store, add some key-value pairs to it, push it to S3 and re-import it from another device or simply sync it over Dropbox. Safe data sharing had never been that simple!

*Secrets are made to be shared, just not with anyone.* Whether you're an admin, a paranoid guy living in a bunker, or a random user who seeks a simple way to store it's critical data in secured manner. *Trousseau* can do something for you.

## Why

Storing, transporting, and sharing sensitive data can be hard, and much more difficult when it comes to automate it.

*Trousseau* was created with private keys and certificates (such as private keys) sharing across a cluster in mind. However it has proved being useful to anyone who need to store and eventually share a passwords store, bank accounts details or even more sensitive data.

---

## Use cases

**For admins and ops** *Trousseau can be useful to you when it comes to:*

- **Store** sensitive data: No more plain certificates and keys in your repositories and configuration files. Your brand new shiny infrastructure surely relies on many certificates and private keys of different kinds: ssl, rsa, gpg, ... *Trousseau* provides a simple and fine-tuned way to store their content in a single file that you can safely version using your favorite version control system.
- **Share** passwords, keys and other critical data with co-workers and servers in your cluster in a safe manner. *Trousseau* can encrypt its content for specific recipients you provide to it (Only the recipient you intend will be able to import and read-write the *Trousseau* store content). *Trousseau* proved itself to be a great way to share some services passwords with your co-workers too! Simply set up a trousseau store with symmetric encryption, sync it over dropbox, et voila!
- **Deploy** keys to your servers in a safe and normative way. Encrypt the trousseau store for each server selectively.

## For the common users

- **Store** your sensitive data such as passwords, bank account details or bitcoin wallets in an encrypted store.
- **Sync** your sensitive data store to remote services and easily share it between your devices.

## How

### Installation

**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:

```
1 $ echo "deb https://dl.bintray.com/oleiade/deb /" | sudo tee /etc/apt/sources.list.d/trousseau.list
```

And you're ready to go:

```
1 $ sudo apt-get update && sudo apt-get install trousseau
```

## OSX

---

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

```
1 brew install oleiade/tap/trousseau
```

**Binaries** Get the latest darwin release zip archive from the repository. Unzip it, and place the trousseau executable wherever it suits you.

```
1 $ unzip trousseau_X.Y.Z_darwin_amd64.zip
2 $ cp trousseau_X.Y.Z_darwin_amd64/trousseau /usr/local/bin
```

### Build it

1. First, make sure you have a Go language compiler **>= 1.5** (*mandatory*) and git installed.
2. Make sure you have the following go system dependencies in your `$PATH`: `bzr`, `svn`, `hg`, `git`
3. Ensure your GOPATH is properly set.
4. run `go build github.com/oleiade/trousseau/cmd/trousseau`
5. The `trousseau` binary is now in your current working directory folder

### Prerequisites

**If you go for OpenPGP asymmetric encryption** 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.

```
1 $ export TROUSSEAU_KEYRING_SERVICE=my_keyring_key
2 $ 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.

---

```
1 $ export GPG_AGENT_INFO=path_to_the_gpg_agent_info_file
2 $ export TROUSSEAU_MASTER_GPG_ID=myid@mymail.com
3 $ trousseau get abc
```

## Whatever encryption style you go for

**Environment variable** You can pass your primary key passphrase as `TROUSSEAU_PASSPHRASE` environment variable:

```
1 $ export TROUSSEAU_PASSPHRASE=mysupperdupperpassphrase
2 $ trousseau get abc
```

**ask-passphrase global option** You can have trousseau asking for your passphrase using the command line global option:

```
1 $ trousseau --ask-passphrase get abc
2 Passphrase:
3 123
```

**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`

## Let's get started

### Basics

### API

**First steps with the data store** First step with **trousseau** is to create a data store.

To do so, you will need to decide the kind of encryption you wish to use: + OpenPGP asymmetric encryption: accessing the data store will be restricted to the recipients (gpg) its been encrypted for. This is probably the best choice if you intend to share the data store with multiple servers or gpg capable devices. It can also be a good choice if you inted to share the data store with a team or selected people. + AES256 symmetric encryption: the data store will be encrypted using a passphrase you

---

will provide. This is probably the best choice if you intend to store sensitive personal informations (passwords, bank details, bitcoins...) and sync it accross devices.

Then, you can proceed and create a data store with the `create` command. As a default: + data stores will be created as `$HOME/.trousseau`. However the global option `store` will allow you to select the place on the filesystem where *trousseau* should create/open the data store. + data stores will be created using asymmetric OpenPGP encryption. However `encryption-type` and `encryption-algorithm` options will allow to select explicitly the encryption mode of your choice.

#### HOWTO

```
1 # create a trousseau for two gpg recipients
2 # both key ids and key email are supported.
3 $ trousseau create 4B7D890,foo@bar.com
4 trousseau created at $HOME/.trousseau
5
6 # Or create a symmetrically encrypted data store
7 # with a passphrase
8 $ trousseau create --encryption-type symmetric
9 Passphrase:
10 trousseau created at $HOME/.trousseau
```

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

```
1 $ cat ~/.trousseau
2 {
3     "crypto_type":1,
4     "crypto_algorithm":0,
5     "_data":"012ue091ido19d81j2d01029dj1029d1029u401294i ... 1028019
6         k0912djm0129d12"
```

If you've just updated *trousseau* to a version marked as implying backward incompatibilities, the `upgrade` command is here to help

```
1 $ trousseau upgrade
2 Upgrading trousseau data store to version M: success
3 Upgrading trousseau data store to version N: success
4 # This is it, your legacy data store has now been upgraded to be
   compatible with
5 # your current version of trousseau
```

### Manipulating keys

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

---

### You've got the keys

```
1 # Right now the store is empty
2 $ trousseau show
3
4
5 # Let's add some data into it
6 $ trousseau set abc 123
7 $ trousseau set "easy as" "do re mi"
8
9 # set action supports a --file flag to use the content
10 # of a file as value
11 $ trousseau set myuser.ssh.public_key --file ~/.ssh/id_rsa.pub
12
13
14 # Now let's make sure data has been added
15 $ trousseau keys
16 abc
17 easy as
18 myuser.ssh.public_key
19
20 # Let's check values too
21 $ trousseau get abc
22 123
23
24 # What about renaming abc key, just for fun?
25 $ trousseau rename abc 'my friend jackson'
26 $ trousseau keys
27 my friend jackson
28 easy as
29 myuser.ssh.public_key
30
31
32 $ trousseau show
33 my friend jackson: 123
34 easy as: do re mi
35 myuser.ssh.public_key: ssh-rsa 1289eu102ij30192u3e0912e
36 ...
37
38 # Whenever you want to export a key value to a file, just use
39 # the get command --file option
40 $ trousseau get myuser.ssh.public_key --file /home/myuser/id_rsa.pub
41
42 # Now if you don't need a key anymore, just drop it.
43 $ trousseau del 'my friend jackson' # Now the song lacks something
    doesn't it?
```

### 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.
  - **rename** KEY\_NAME NEW\_NAME : Renames a store key
  - **del** KEY : Deletes provided key from the store
  - **keys** : Lists the stored keys
  - **show** : Lists the stored key-value pairs

## Importing/Exporting to remote storage

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, SSH and gist storages are available (more are to come).

## API

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

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

```
1 {protocol}://{identifier}:{secret}@{host}:{port}/{path}
```

- **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.

### S3 Example

```
1 # Considering a non empty trousseau data store
2 $ trousseau show
3 abc: 123
4 easy as: do re mi
5
6 # And then you're ready to push
7 $ trousseau push s3://aws_access_key:aws_secret_key@bucket:region/
  remote_file_path
```

---

```
8
9
10 # Now that data store is pushed to S3, let's remove the
11 # local data store and pull it once again to ensure it worked
12 $ rm ~/.trousseau
13 $ trousseau show
14 Trousseau unconfigured: no data store
15
16 $ trousseau pull s3://aws_access_key:aws_secret_key@bucket:region/
    remote_file_path
17 $ trousseau show
18 abc: 123
19 easy as: do re mi
```

### Scp example

```
1 # We start with a non-empty trousseau data store
2 $ trousseau show
3 abc: 123
4 easy as: do re mi
5
6 # To push it using scp we need to provide it a couple of
7 # basic options.
8 # Nota: In order for your remote password not to appear
9 # in your shell history, we strongly advise you to use
10 # the push/pull --ask-password option instead of supplying
11 # the password through the dsn.
12 $ trousseau push --ask-password scp://user:@host:port/remote_file_path
13 Password:
14 Trousseau data store succesfully pushed to ssh remote storage
15
16
17 # Now that data store has been pushed to the remote storage
18 # using scp, let's remove the local data store and pull it
19 # once again to ensure it worked
20 $ rm ~/.trousseau
21 $ trousseau show
22 Trousseau unconfigured: no data store
23
24 $ trousseau pull --ask-password scp://user:@host:port/remote_file_path
25 Password:
26 Trousseau data store succesfully pulled from ssh remote storage
27
28 $ trousseau show
29 abc: 123
30 easy as: do re mi
```

**Gist example** To use the gist remote storage support, you will need to generate a Github personal access token. Once you've generated one, use it as the dsn *password* field as in the following exam-



---

ple:

```
1 # We start with a non-empty trousseau data store
2 $ trousseau show
3 abc: 123
4 easy as: do re mi
5
6 # Nota:
7 # * Gist remote storage doesn't use the host and port dsn fields,
8 #   but you still need to provide their ':' separator
9 $ trousseau push gist://user:mysuppedupertoken@:/gist_name
10 Password:
11 Trousseau data store succesfully pushed to gist remote storage
12
13
14 # Now that data store has been pushed to gist
15 # let's remove the local data store and pull it
16 # once again to ensure it worked
17 $ rm ~/.trousseau
18 $ trousseau show
19 Trousseau unconfigured: no data store
20
21 $ trousseau pull gist://user:mysupperduppertoken@:/gist_name
22 Password:
23 Trousseau data store succesfully pulled from gist
24
25 $ trousseau show
26 abc: 123
27 easy as: do re mi
```

## Local imports and exports

### 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.

### Real world example

```
1 $ trousseau export testtrousseau.asc # Fine we've exported our current
   data store into a single file
2 $ mail -f testtrousseau.asc cousin@machin.com # Let's pretend we've
   sent it by mail
3
4 # Now cousin machin is now able to import the data store
5 $ trousseau import testtrousseau.asc
6 $ trousseau show
7 cousin_machin:isagreatbuddy
```

---

```
8 adams_family:rests in peace, for sure
```

## Metadata

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

```
1 $ trousseau meta
2 CreatedAt: 2013-08-12 08:00:20.457477714 +0200 CEST
3 LastModifiedAt: 2013-08-12 08:00:20.457586991 +0200 CEST
4 Recipients: [4B7D890,28EA78B]
5 TrousseauVersion: 0.1.0c
```

## (OpenPGP encryption) 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.

```
1 $ trousseau add-recipient 75FE3AB
2 $ trousseau add-recipient 869FA4A
3 $ trousseau meta
4 CreatedAt: 2013-08-12 08:00:20.457477714 +0200 CEST
5 LastModifiedAt: 2013-08-12 08:00:20.457586991 +0200 CEST
6 Recipients: [4B7D890, 75FE3AB, 869FA4A]
7 TrousseauVersion: 0.1.0c
8
9 $ trousseau remove-recipient 75FE3AB
10 $ trousseau meta
11 CreatedAt: 2013-08-12 08:00:20.457477714 +0200 CEST
12 LastModifiedAt: 2013-08-12 08:00:20.457586991 +0200 CEST
13 Recipients: [4B7D890, 869FA4A]
14 TrousseauVersion: 0.1.0c
```

## Contribute

### Testing

Running the unit tests for the project is done with the following command:

```
1 go test github.com/oleiade/trousseau/...
```

---

## Workflow

For detailed contribution instructions, see the the CONTRIBUTING document

However here is a quick summary for all of you in a hurry: \* Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug. \* Fork the repository on GitHub. \* Start a branch from **develop** dedicated to your changes. \* If relevant: write tests showing the bug was fixed or the feature implemented works as expected. \* Send a pull request and bug the maintainer until it gets merged and published. :) Make sure to add yourself to AUTHORS.

## It's open-source

*Trousseau* is open source software under the MIT license. Any hackers are welcome to supply ideas, features requests, patches, pull requests and so on. Let's make *Trousseau* awesome!

See **Contribute** section.

## Changelog

See CHANGELOG