

---

# **sett - secure encryption and transfer tool**

***Release 4.4.3***

**Gerhard Bräunlich,  
Christian Ribeaud,  
Jarosław Surkont,**

**Robin Engler,  
Kevin Sayers,  
François Martin**

**Apr 23, 2024**



# TABLE OF CONTENTS

<b>1</b>	<b>What is sett?</b>	<b>3</b>
<b>2</b>	<b>sett quick start guide</b>	<b>5</b>
2.1	Initial setup . . . . .	5
2.2	Getting a recipient's PGP key . . . . .	5
2.3	<i>sett-gui</i> quick start . . . . .	6
2.3.1	Encrypting data . . . . .	6
2.3.2	Transferring data . . . . .	6
2.3.3	Decrypting data . . . . .	7
2.4	sett command line quick start . . . . .	7
<b>3</b>	<b>Installing sett on your local computer</b>	<b>9</b>
3.1	Installing dependencies . . . . .	9
3.1.1	Installing dependencies on Windows . . . . .	9
3.1.2	Installing dependencies on Linux . . . . .	11
3.1.3	Installing dependencies on Mac OS . . . . .	11
3.2	Installing sett . . . . .	11
3.2.1	Installing sett on Windows . . . . .	12
3.2.2	Installing sett on Linux . . . . .	12
3.2.3	Installing sett on Mac OS . . . . .	13
3.2.4	Installing sett in a Conda virtual environment [cross-platform] . . . . .	13
3.2.5	Installing a specific version of sett . . . . .	13
3.3	Updating sett . . . . .	14
<b>4</b>	<b>PGP key management with sett</b>	<b>15</b>
4.1	Generate a new public/private PGP key pair . . . . .	15
4.1.1	Generate a new key pair with <i>sett-gui</i> . . . . .	15
4.1.2	Generate a new key pair in command line . . . . .	17
4.1.3	List your local PGP keys in command line . . . . .	18
4.1.4	Export/import private keys in command line . . . . .	18
4.2	Upload your public PGP key to the keyserver . . . . .	19
4.2.1	Upload your public PGP key with <i>sett-gui</i> . . . . .	19
4.2.2	Upload your public PGP key in command line . . . . .	19
4.2.3	Upload your public PGP key using a web browser . . . . .	20
4.3	Register your public PGP key in the BioMedIT portal . . . . .	22
4.3.1	PGP key status . . . . .	22
4.3.2	Register your public PGP key . . . . .	23
4.4	Download public PGP keys from the default keyserver . . . . .	24
4.4.1	Download PGP keys with <i>sett-gui</i> . . . . .	24
4.4.2	Download PGP keys in command line . . . . .	25
4.4.3	Remove your public PGP keys from the keys.openpgp.org keyserver . . . . .	25
4.5	Delete PGP keys from your local machine . . . . .	26
4.6	Revoke your PGP key . . . . .	26
4.6.1	Revoke your PGP key with <i>sett-gui</i> . . . . .	26

4.6.2	Revoke your PGP key in command line . . . . .	27
4.7	Migrating keys from the GnuPG keyring to the sett certificate store . . . . .	27
4.8	Introduction to public-key cryptography, PGP and GnuPG . . . . .	28
4.8.1	Key fingerprints . . . . .	29
4.8.2	File encryption . . . . .	29
4.8.3	File signing . . . . .	29
4.8.4	Revocation certificates . . . . .	30
4.8.5	Exchanging public keys via a keyserver . . . . .	30
4.9	Troubleshoot the “Network is unreachable” and “No keyserver available” error . . . . .	31
<b>5</b>	<b>Generating SSH keys</b>	<b>33</b>
5.1	Windows users: enabling the ssh-keygen command . . . . .	34
<b>6</b>	<b>Encrypting, transferring and decrypting data with sett</b>	<b>35</b>
6.1	Encrypting files . . . . .	35
6.1.1	Output file naming scheme . . . . .	36
6.1.2	Encrypting data with sett-gui . . . . .	36
6.1.3	Encrypting data on the command line . . . . .	39
6.2	Transferring files . . . . .	40
6.2.1	Transferring files with sett-gui . . . . .	41
6.2.2	Transferring data on the command line . . . . .	42
6.3	Decrypting files . . . . .	43
6.3.1	Decrypting data with sett-gui . . . . .	44
6.3.2	Decrypting data on the command line . . . . .	45
6.4	PGP key auto-download and auto-refresh in <i>sett</i> . . . . .	45
6.5	Setting up predefined connection profiles for frequent use . . . . .	46
6.6	Working with large files . . . . .	46
6.6.1	Split and transfer large files . . . . .	47
6.7	Automating tasks with sett . . . . .	48
6.8	Installing or running sett behind a proxy . . . . .	48
6.8.1	Checking whether you are using a proxy . . . . .	49
6.9	Known issues and limitations . . . . .	49
6.9.1	SSH private key with non-ASCII characters password . . . . .	49
<b>7</b>	<b>Automating sett</b>	<b>51</b>
7.1	Batch packaging and transferring of files . . . . .	51
7.2	Dealing with passwords when automating sett tasks . . . . .	51
7.2.1	Example 1: Fully automated - unencrypted password file . . . . .	51
7.2.2	Example 2: Semi automated . . . . .	52
<b>8</b>	<b>sett configuration file</b>	<b>53</b>
8.1	Configuration changes using <i>sett-gui</i> . . . . .	53
8.2	Configuration changes using command line . . . . .	55
8.3	Alternative config file location . . . . .	55
8.4	Configuration options . . . . .	55
<b>9</b>	<b>sett packaging file format specifications</b>	<b>59</b>
9.1	File structure . . . . .	59
9.2	File example . . . . .	60
<b>10</b>	<b>sett benchmarks</b>	<b>61</b>
10.1	Setting-up your system to run the sett benchmarks . . . . .	61
10.1.1	Requirements . . . . .	61
10.2	Running the sett benchmarks . . . . .	61
10.3	Benchmark results plotting . . . . .	62
10.4	Benchmark parameters . . . . .	63
10.4.1	Benchmark factor levels . . . . .	63
10.4.2	Input/Output file locations . . . . .	64
10.4.3	SFTP parameters . . . . .	64

10.4.4	Misc parameters . . . . .	65
10.5	Benchmark results and sett performance . . . . .	65
10.5.1	Overall sett performance . . . . .	65
10.5.2	Compression level comparison . . . . .	66
<b>11</b>	<b>Frequently asked questions and bug reports</b>	<b>71</b>
11.1	Bug reports . . . . .	71
11.2	Frequently asked questions . . . . .	71
11.2.1	Warning “Config: Unexpected keys: ...” at startup . . . . .	71
11.2.2	Nothing happens when running <i>sett-gui</i> (Windows) . . . . .	71
11.2.3	Error “No module named ‘PySide2’” on <i>sett</i> startup (Linux / MacOS) . . . . .	72
11.2.4	sett generates SHA-1 keys . . . . .	72
<b>12</b>	<b>Source code</b>	<b>73</b>



**Warning:** This version of **sett** and its associated documentation are deprecated and will be retired by the **end of June 2024**. This documentation remains available for reference only.

The new version of **sett** can be found at <https://gitlab.com/biomedit/sett-rs>, and the associated documentation is hosted [here](#)). We strongly encourage all users to migrate to the new version for improved performance and security.

Welcome to the official **sett** documentation page. Please use the table of contents menu to the left or below to navigate the topics.

---

**Important: IMPORTANT NOTICE - PLEASE READ CAREFULLY**

Starting with **version 4.4.0**, *sett* has been updated to no longer use GnuPG as encryption backend by default.

**WHAT YOU NEED TO DO:**

- **If you are a new *sett* user** with no PGP key (the cryptographic key used for data encryption, decryption and signing), you can ignore this message.
- **If you were already using earlier *sett* versions**, you will need to migrate the PGP key(s) you are using with *sett* to the new certificate store. Please refer to the instructions given in the section *Migrating keys from the GnuPG keyring to the sett certificate store*.

**WHAT ELSE CHANGES FOR YOU AS A USER:**

- After you migrated your key(s) to the new *sett* certificate store, nothing else changes - simply keep using *sett* as you did before.

**MORE DETAILS ABOUT THIS CHANGE**

- Starting with version 4.4.0, *sett* no longer uses GnuPG as its default cryptographic backend. Instead, it now uses an embedded cryptographic library named *sequoia* <<https://sequoia-pgp.org>> and its own keystore.
- As a result, the Open PGP keys located in your GnuPG keyring are no longer automatically detected by *sett*, and they must therefore be migrated to the new *sett* certificate store.
- It remains possible to switch back to using GnuPG as cryptographic backend by enabling the `legacy_mode` option in the *configuration file* or “Use GnuPG legacy mode” in the settings tab of *sett-gui*.





## WHAT IS SETT?

**sett** stands for “Secure Encryption and Transfer Tool” and is a python-based wrapper around GnuPG and SFTP that automates the packaging, encryption, and transfer of data.

**sett** is available both as a GUI program `sett-gui`, and in command line `sett`. Most functionality of *sett* is available in both the GUI and the command line.

**sett** is developed as part of the [BioMedIT](#) project. It is licensed under the GPLv3 (GNU General Public License) and the source code is available from its public [GitLab repo](#)

---

### BioMedIT

When using **sett** to transfer data into the **SPHN BioMedIT network**, a number of additional constraints apply. These specific instructions are highlighted throughout this documentation in **BioMedIT labeled boxes**, just **like this one**.

---



## SETT QUICK START GUIDE

### 2.1 Initial setup

1. Install *sett* following the instructions given in *Installing sett on your local computer*.
2. Run `sett-gui`.
3. If you do not already possess a private/public PGP key pair, go to the **Keys** tab and create one following the instructions given in the *Generate a new public/private PGP key pair* section. You should see your new key listed in the **Private keys** and **Public keys** fields of the **Keys** tab.

---

#### BioMedIT

Your PGP key must be registered with the [BioMedIT portal](#) before it can be used to encrypt, decrypt, or sign data packages. Please see the *Register your public PGP key in the BioMedIT portal* section for details.

---

4. If not already done, download the public PGP key of the recipient(s) to whom you intend to send data (or from whom you will receive data). In *sett-gui*, go to the **Keys** tab and click on the **Download keys** icon, then search for your contact's key by either entering the email or fingerprint associated with your contact's PGP key.
5. Just after downloading you contact's PGP key, verify it to make sure that it is genuine. This can be done by either:
  - If you are a BioMedIT user: in *sett-gui*, select your contact's key in the **Public keys** list (under the **Keys** tab) and check that its status is approved by looking for the green text: "This key is approved".
  - Alternatively, contact the key owner and verify the key **fingerprint** with them.

### 2.2 Getting a recipient's PGP key

In order to encrypt data for a given recipient, you must first obtain a copy of their public PGP key. Public PGP keys are non-sensitive and can be freely exchanged.

1. Go to the **Keys** tab of the *sett* interface and click on the **Download key** icon.
2. A pop-up will appear, where you can search your recipient's key by email or key fingerprint.

---

#### BioMedIT

**Please make sure that your recipient's key is approved.** A green text "This key is approved" should be displayed when the key is selected in the **Keys tab** of *sett-gui*. Only approved keys can be used within BioMedIT.

---

## 2.3 *sett-gui* quick start

### 2.3.1 Encrypting data

1. Go to the **Encrypt** tab of the *sett* interface.
2. Add one or more files and directories to encrypt by clicking the **Add files** or **Add directories** icons.
3. Select your own PGP key in the **Sender** field. This is the key that will be used to sign the encrypted data.
4. Add one or more **Recipients** by selecting them in the drop-down menu and clicking on **+**. These are the keys that will be used to encrypt the data, i.e. only these recipients will be able to decrypt the data.

---

#### BioMedIT

The selected **Recipients** must be approved **Data Managers** of the project for which data is being encrypted.

---

5. **DTR ID:** specifying a valid **Data Transfer Request ID** is mandatory when a data package is transferred into the BioMedIT network. For other destinations, the **DTR ID** field can be left empty (or set to any arbitrary value), and the **Verify DTR** checkbox must be disabled (in the **Settings** tab).

---

#### BioMedIT

The **Verify DTR** checkbox should always be enabled, since a valid **DTR ID** is required by the BioMedIT network.

---

6. **Purpose:** purpose of the data transfer, please select either PRODUCTION or TEST, or leave it empty.

---

#### BioMedIT

This field is mandatory.

---

7. **Suffix** and **Location:** optionally, a custom suffix can be specified and will be appended to the file name. The location where the output file should be saved can also be specified.
8. **Compression level** slider: select the amount of compression to apply to the input data when packaging it. Compression values range between 0 (no compression) and 9 (highest compression). Higher compression level result in smaller encrypted output files but require more compute time. If compression is not required, e.g. because the input data is already in a compressed form, then the compression level should be set to 0 in order to speed-up the packaging task.
9. Click **Package & Encrypt** to run the encryption workflow on your data.

### 2.3.2 Transferring data

1. Go to the **Transfer** tab of the *sett* interface.
2. Select one or more files to transfer using the **Add files** button.
3. Select the transfer **Protocol** to be used (**sftp**, **s3** or **liquid files**).
4. Enter the required transfer **Parameters** information depending on the selected protocol.

---

#### BioMedIT

For transfers into the BioMedIT network the transfer protocol and associated parameters are provided by your local BioMedIT node.

---

5. Click **Transfer selected files** to start transferring your files.

### 2.3.3 Decrypting data

1. Go to the **Decrypt** tab of the *sett* interface.
2. Select one or more files to decrypt using the **Add files** button.
3. Specify your desired **Output location**.
4. Click on **Decrypt selected files**.

## 2.4 sett command line quick start

The main commands to encrypt, transfer and decrypt data with *sett* CLI are given here. Note that key management is not implemented in the CLI. Please use *sett-gui* or the *gpg* command.

```
# Data encryption:
sett encrypt --sender <sender key fingerprint or email> --recipient <recipient key_
↳fingerprint or email> --dtr-id <data transfer ID> --purpose <purpose> --output
↳<output file/directory name> <files or directories to encrypt>

# Data transfer.
# to SFTP server:
sett transfer --protocol sftp --protocol-args '{"host": "HOST", "username":"USERNAME",
↳ "destination_dir":"DIR", "pkey":"PRIVATE_RSA_SSH_KEY"}' <files to transfer>
# to S3 server ('session_token' is STS credentials specific):
sett transfer --protocol s3 --protocol-args '{"host": "localhost:9000", "secure":_
↳false, "bucket":"BUCKET", "access_key":"ACCESS_KEY", "secret_key":"SECRET_KEY",
↳"session_token":"SESSION_TOKEN"}' <files to transfer>
# to liquid-files server:
sett transfer --protocol liquid_files --protocol-args '{"host": "HOST", "subject":
↳"SUBJECT", "message": "MESSAGE", "api_key":"APIKEY", "chunk_size": 100}' <files to_
↳transfer>

# Data decryption:
sett decrypt --output_dir <output directory> <files to decrypt>
```



## INSTALLING SETT ON YOUR LOCAL COMPUTER

The Secure Encryption and Transfer Tool - or *sett* runs on all major operating systems. Please refer to the relevant section below depending on your operating system.

*sett* requires the following dependencies to be installed on your system:

- `python`  $\geq 3.8$

Additionally, *sett* also requires users to have:

- A public/private `PGP key pair`. See *PGP key management with sett* for detailed instructions.
- The public PGP key of users for whom they want to encrypt data (i.e. users to whom they want to send data).
- For users who intend to transfer data via SFTP, a `SSH key`. See *Generating SSH keys* for detailed instructions.

*Note:* running *sett* in **legacy mode** additionally requires the user to have `GPG (Gnu Privacy Guard)`  $\geq 2.2.8$  installed on their system.

### 3.1 Installing dependencies

#### 3.1.1 Installing dependencies on Windows

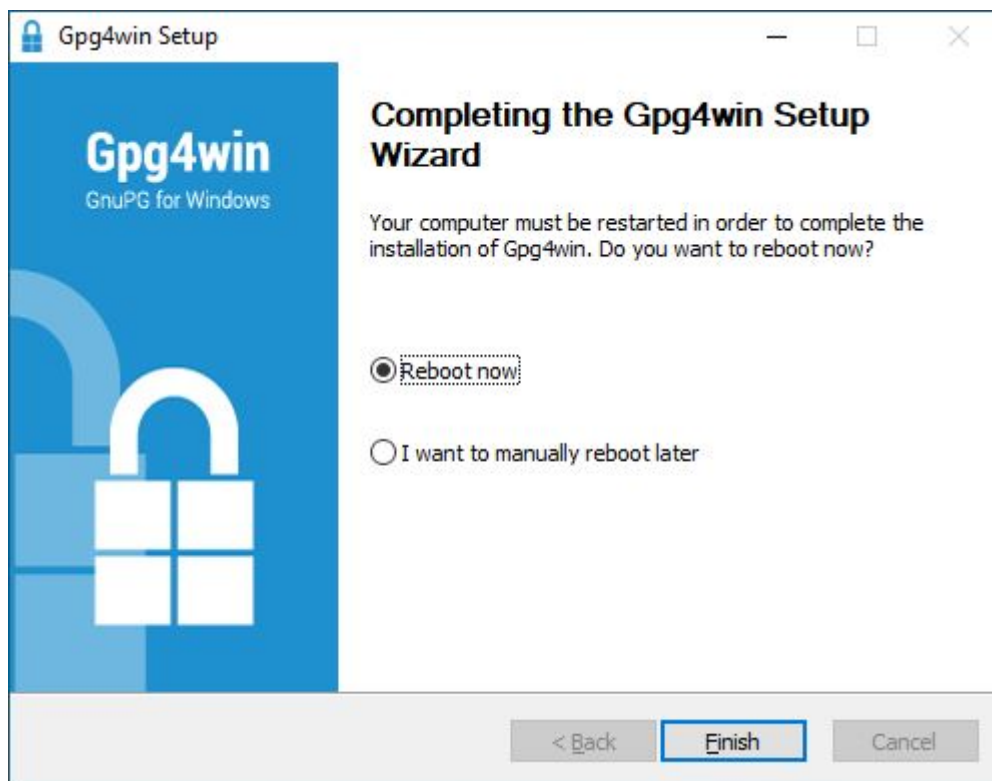
1. Download the installer for the latest python 3 version from <https://www.python.org>.
2. Run the downloaded installer file. In the installation wizard, make sure to select the **Add Python to PATH** option, as shown in the figure below.



3. Reboot your machine.

**Installing optional dependency GnuPG:** this is only required when running *sett* in **legacy mode**:

1. Download the latest version of GPG for windows (gpg4win) from <https://www.gpg4win.org>.
2. Run the installer and accept all default settings (keep clicking “next”). At the end of the installation, a prompt might ask to to reboot the operating system to complete the install.



3. Reboot your machine.



### 3.1.2 Installing dependencies on Linux

Python 3 can be installed from most Linux distributions' software repositories. Here are commands for some of the most popular distributions:

- Ubuntu, Debian: `sudo apt-get install python3 python3-pip`
- Fedora: `sudo dnf install python3 python3-pip`
- CentOS 7: `sudo yum install python36 python36-pip`

**Installing optional dependency GnuPG:** this is only required when running *sett* in **legacy mode**:

- Ubuntu, Debian: `sudo apt-get install gnupg2`
- Fedora: `sudo dnf install gnupg2`
- CentOS 7: `sudo yum install gnupg2`

### 3.1.3 Installing dependencies on Mac OS

In most cases, Python will be pre-installed on Mac OS systems. To check whether it's installed or not, go to **Applications > Utilities** and click on **Terminal**. Alternatively, click **command + spacebar** on the keyboard, type **terminal**, and then click **Enter**.

A terminal window should now have opened. To find out which version of Python is installed, use the following commands:

```
# Check version of default python:
$ python --version
Python 2.7.16

# Check whether python3 is installed:
$ python3 --version
Python 3.X.X
```

**Installing Python 3 and GnuPG:** the easiest way to install python 3 (if needed) is using **Homebrew** - <https://brew.sh>. Instructions to install Homebrew can be found on its website. Note that a prerequisite for Homebrew is Apple's xcode tools (`xcode-select --install`). With Homebrew available on your system, installing Python 3 is now a one-liner:

```
$ brew install python3
```

**Installing optional dependency GnuPG:** this is only required when running *sett* in **legacy mode**:

```
$ brew install gpg
```

## 3.2 Installing sett

---

**Note:** On some older operating systems (e.g. CentOS 7) *sett-gui* might not run due to the outdated system libraries. In such cases it is necessary to run `pip install sett[legacy]` instead of `pip install sett` (see the instructions below).

---

---

**Note:** If you encounter any error during the installation or when trying to start *sett*, please have a look at [the FAQ section](#) to see whether your problem does already have an answer.

---

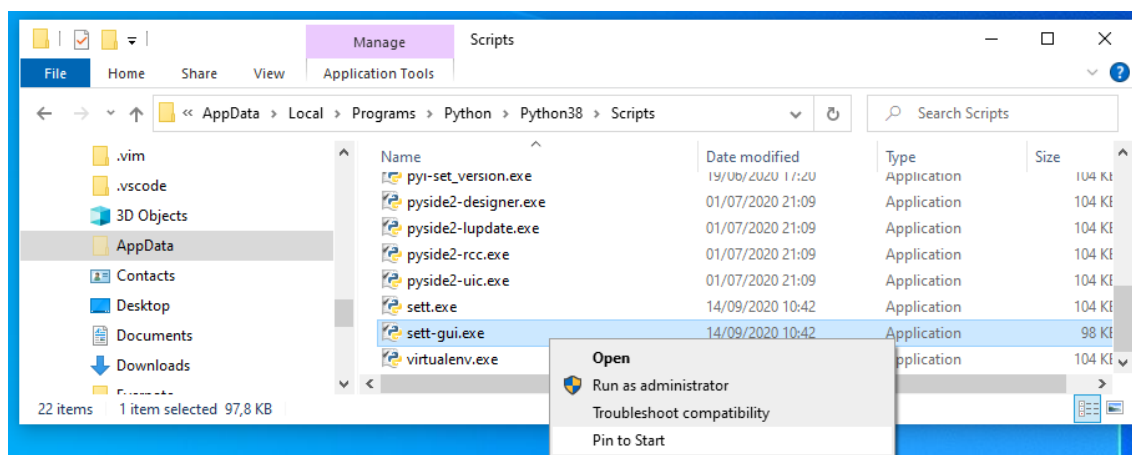
### 3.2.1 Installing sett on Windows

1. Start a Windows shell by pressing the Windows key, then type `cmd` on your keyboard and press enter.
2. In the Windows shell, run the following command: `pip install sett`. If you are accessing the internet via a proxy, please refer to [this section](#) for further instructions.
3. To check that `sett` has been correctly installed, try to type: `sett --help` in the terminal. The `sett` help should be displayed.
4. To start the `sett` application in graphical mode, type `sett-gui` in the Windows terminal.

Alternatively, depending on the Python version installed, an executable can be found under C:\Users\%username%\AppData\Local\Programs\Python\Python38\Scripts\sett-gui.exe or C:\Users\%username%\AppData\Roaming\Python\Python310\Scripts\sett-gui.exe or a similar location.

Note that to navigate to this location using the Windows explorer, you might have to enable **show hidden files** in the Windows explorer menu. Alternatively you can copy/paste `C:\Users\%username%\AppData\Local\Programs\Python\Python38\Scripts` or `C:\Users\%username%\AppData\Roaming\Python\Python310\Scripts` in the Windows explorer bar.

For convenience, a shortcut to *sett* can be added to the Windows start menu: right-click on the `sett-gui.exe` file, then select **Pin to Start**. *sett* can now be run directly from the Windows start menu.



### 3.2.2 Installing sett on Linux

With python 3 installed on your machine, the *sett* application can be installed by typing the following command in the terminal:

```
pip install --user sett
```

If you are accessing the internet via a proxy, please refer to [this section](#) for further instructions.

**Note:** If you have both python 2 and python 3 installed on your machine, you may have to use `pip3` instead of `pip` in the commands above.

**Note:** Without the `--user` option, the above instruction will install the python scripts into system space, which will require administrator privileges. Adding `--user`, installs the module into a platform dependent user directory (e.g. `~/local/bin/` in the case of Linux).

To check that *sett* has been correctly installed, type `sett --help` in the terminal. The *sett* help should be displayed. To start the app in graphical mode, type `sett-gui` in the terminal.

### 3.2.3 Installing sett on Mac OS

Follow the same installation instructions as found in the Linux section above.

If you experience SSL certification error messages when running *sett*, such as:

```
certificate verify failed: unable to get local issuer certificate.
```

or:

```
Could not connect to pypi repository to query the latest version.
```

you will need to execute the file `Install Certificates.command`, found in your Python 3.X application directory. Typically the file is located in `~/Applications/Python 3.X/Install Certificates.command`.

**Note:** In case of a system wide python installation, the `Install Certificates.command` file will be located in `/Applications/Python 3.X/Install Certificates.command`.

In this scenario, running the file will also require root privileges.

### 3.2.4 Installing sett in a Conda virtual environment [cross-platform]

The Conda package manager can be used to create a virtual, sandboxed, environment in which *sett* can then be installed:

```
conda create -y --name sett_venv --no-default-packages python=3.8
conda activate sett_venv
pip install sett

sett -h           # Display sett help to check it was properly installed.
sett-gui          # Start sett in graphical mode.
conda deactivate  # When you are done using sett, close the conda environment.
```

**Removing the conda environment:** to delete the Conda environment where *sett* is installed, the following command can be used:

```
conda env remove --name sett_venv
```

**Change install location:** by default, Conda installs the `sett_venv` environment into the user's home directory. To specify an alternative location, replace `--name sett_venv` with `--prefix /alternative/path/sett_venv`, as shown in this example:

```
conda create -y --prefix /alternative/path/sett_venv --no-default-packages python=3.8
conda activate /alternative/path/sett_venv
conda env remove --prefix /alternative/path/sett_venv
```

### 3.2.5 Installing a specific version of sett

Specific versions of *sett* can be download from <https://gitlab.com/biomedit/sett/-/releases>. After having unpacked the downloaded file and changed the working directory into the root of the unpacked files, the installation can be done with the command:

```
pip install --user .
```

For proxy settings please refer to [this section](#).

### 3.3 Updating sett

As only the latest version of *sett* is officially supported and guaranteed to work, it is strongly recommended to always keep your local installation of *sett* up-to-date.

In most cases, the following command can be used to upgrade *sett*:

```
pip install --upgrade --user sett
```

For proxy settings please refer to [this section](#).

---

**Note:** If you have both python 2 and python 3 installed on your machine, you may have to use `pip3` instead of `pip` in the command above.

If you have installed *sett* in a Conda virtual environment, make sure the environment is activated before running the upgrade command.

---

If you have not installed *sett* yourself, please kindly ask your administrator to perform the upgrade for you.

## PGP KEY MANAGEMENT WITH SETT

To encrypt and decrypt data, *sett* uses **public key cryptography**. If you are not familiar with public key cryptography concepts such as **public and private** keys, **revocation certificates**, or **keyservers** you are advised to read *this introductory section*.

### 4.1 Generate a new public/private PGP key pair

A prerequisite to encrypt, decrypt and transfer files with *sett* is to have a public/private PGP key pair.

---

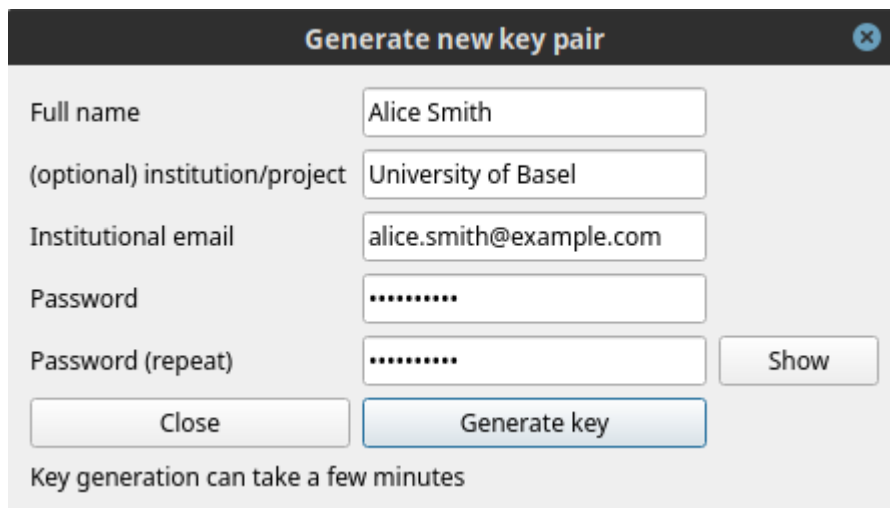
**Important:** Always keep your **private key**, its **password**, and its **revocation certificate** in a secure location. Never share any of this information with anyone.

---

#### 4.1.1 Generate a new key pair with sett-gui

To generate a new public/private key pair using the *sett* graphical user interface:

1. Go to the **Keys** tab and click **Generate new private/public key**. A dialog box will appear.



Generate new key pair

Full name: Alice Smith

(optional) institution/project: University of Basel

Institutional email: alice.smith@example.com

Password: .....

Password (repeat): ..... Show

Close Generate key

Key generation can take a few minutes

2. Fill-in the required fields:
  - **Full name:** key owner first and last name.
  - **Institution/project:** optionally, enter the institution you are affiliated with.
  - **Institutional email:** enter the email to be associated with the key pair.
  - **Password:** the password must be at least 10 characters long, and it is highly recommended that it contain a mix of letters, numbers and special characters.

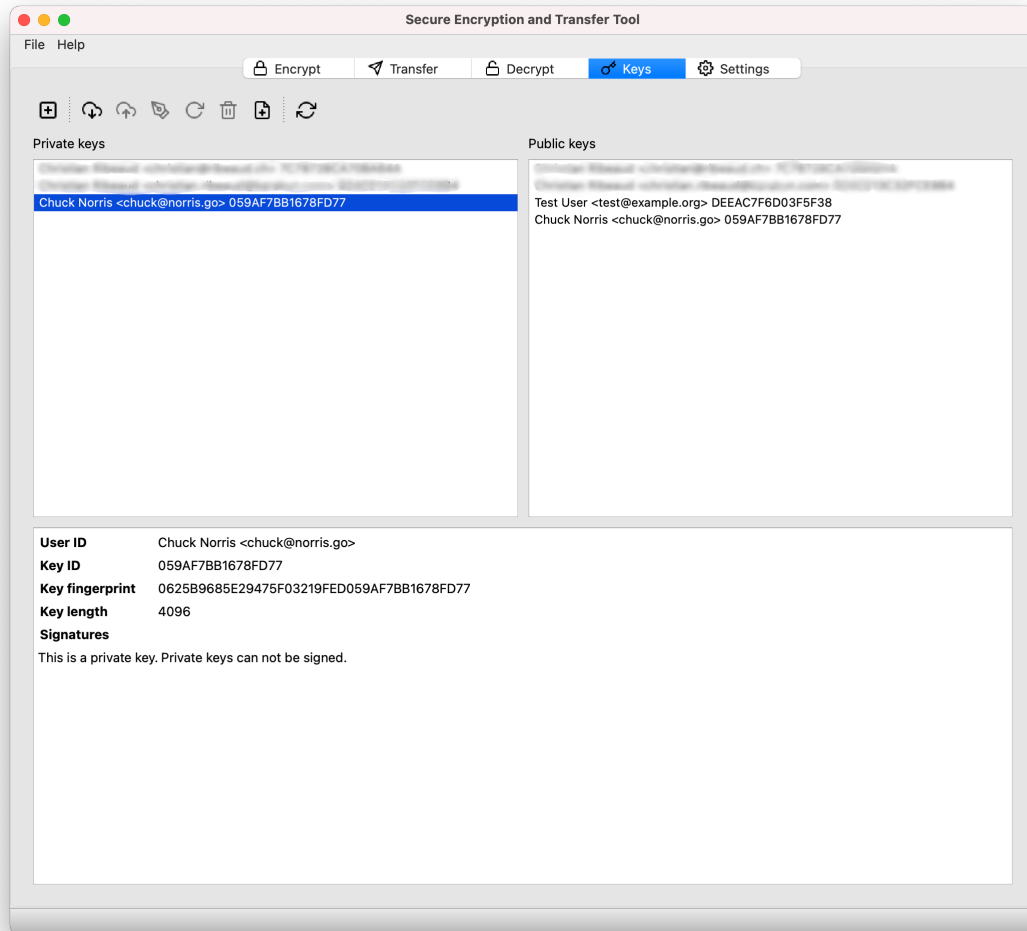
**Warning:** It is **not possible to retrieve a password from a generated key**, so make sure to remember it and store it in a safe place (e.g. a password manager).

3. Click **Generate key** and wait for a few seconds, a new key pair will be created.
4. A new pop-up window will appear to confirm that the key was generated successfully. Click **Show details** to display the revocation certificate associated with the new key, as shown in the figure below.

Copy the revocation certificate to a safe location, ideally a password manager, and click **OK** to close the dialog box.



5. The new key should now be listed in *sett* under both **Private keys** and **Public keys**.



6. Now that your new PGP key is created, make sure to *Upload it to the keyserver*, and to *register it in the BioMedIT portal*.

### 4.1.2 Generate a new key pair in command line

The *sett* command line interface does not implement new key generation. Please use the GnuPG command line instead:

```
gpg --full-generate-key
```

During the key generation process, a number of questions must be answered. Here are the suggested settings:

```
type of key      : (1) RSA and RSA (default)
key bit length   : 4096
key validity time: 0 (Key does not expire)
Real name        : your full name (e.g. Alice Smith)
Email address:    : your institutional email address, will be associated
                   : permanently to your key.
Comment          : optionally, a "comment" (e.g. the institution you work
                   : for) can be added to the key metadata.
```

**Important:** At the end of the key generation process, *gpg* will ask for a pass phrase (password) that will be associated with the private key. Please use a reasonably strong password and make sure to save it in a password

manager.

**If the password is forgotten, there is no way to retrieve or re-create it.** Your PGP key pair will thus become unusable and will need to be revoked.

---

After having created a new key, it is strongly recommended to immediately generate a revocation certificate. Note that the password of the private key must be entered in order to complete the commands below.

```
# General syntax:
gpg --output <revocation certificate file> --gen-revoke <key fingerprint>

# Example:
gpg --output revocation_certificate.asc --gen-revoke_
↪AD1008B5F0F876630B9AF245AA2BF155EF144550
```

### 4.1.3 List your local PGP keys in command line

Private and public keys available in the local keyrings can be listed with the following commands:

```
gpg --list-keys          # list public key(s).
gpg --list-secret-keys  # list private key(s).
```

### 4.1.4 Export/import private keys in command line

In some situations (new computer setup, remote *sett* environment, ...) you might need to copy or move your private key to a different machine. This can be done by **exporting the private key** to a file, transferring the file to the new machine, and **importing** it.

Here is an example of how to export a private key to a file with GnuPG:

```
# The key identifier can be its fingerprint, key ID or the user's email address.
# NOTE: the order of arguments matters, the '--output' option must be passed first!
gpg --output private_key.gpg --export-secret-key username@email
```

You will be prompted for the key password. Note that keys can also be exported in ASCII format (instead of binary) by adding the `--armor` option.

After the key has been transferred to the new machine (in either binary or ASCII format), importing it is as easy as:

```
gpg --import private.gpg
```

Verify that the key has been correctly imported with `gpg --list-secret-keys`.

**Warning:** the above method exports only a single key, not the entire content of your GnuPG keyring. If your intention is to create a backup of all your PGP keys, follow the instructions given [here](#) or have a look at this [procedure](#) instead.

Ensure that you store any backed-up secret keys in a secure location and in an encrypted form (typically in a password manager).



## 4.2 Upload your public PGP key to the keyserver

*sett* allows users to upload their public PGP key to the keyserver specified in its *configuration file*.

### BioMedIT

By default, a new *sett* installation is configured to upload keys to the [OpenPGP keyserver](https://keys.openpgp.org). If you have used *sett* before, make sure your keyserver setting is no longer pointing to “keyserver.dcc.sib.swiss”, as this has now been decommissioned.

[keys.openpgp.org](https://keys.openpgp.org) is a public keyserver for the distribution and discovery of public PGP keys. This service is GDPR-compliant, based on the open source software [Hagrid](https://hagrid.org), and already used by over 300'000 users worldwide. For more information about the service, please see [this link](#).

### 4.2.1 Upload your public PGP key with sett-gui

Uploading your public PGP key to the keyserver specified in *sett* is straightforward:

1. In the **Settings** tab, make sure that the **Keyserver URL** points to <https://keys.openpgp.org>.
2. In the **Keys** tab, select your public PGP key from the **Public keys** list.
3. Click on the **Upload selected keys to keyserver** icon (arrow pointing up into cloud).
4. A dialog box will appear to ask for confirmation.
5. To **verify your key** with the keyserver (i.e. associate your email address with your key on the keyserver), make sure that [x] **Associate the key with your identity is checked**. In practical terms, this means that people will now be able to search for your key on the keyserver using your email address - otherwise your key will only be searchable using its full fingerprint.
6. Click **OK**.
7. If you have selected the checkbox, **you will shortly receive an email** from [keyserver@keys.openpgp.org](mailto:keyserver@keys.openpgp.org) that contains a confirmation link to prove that you have access to your email.
8. Open the email and **click on the link** to associate your key with your email address. Other users now can find your public key with your email address.

### 4.2.2 Upload your public PGP key in command line

Uploading keys via the command line can be done either using *sett* or *GnuPG*. In both cases, you first need to **retrieve the fingerprint associated with your PGP key**. The fingerprint is a 40 characters-long hexadecimal string. This can be done with the following command:

```
# Using a recent GnuPG version [≥ 2.2.x]:
gpg --list-secret-keys

# Using older GnuPG versions: make sure to remove the spaces present in
# the displayed fingerprint (a fingerprint is a 40 characters-long string
# with no spaces).
gpg --list-secret-keys --with-fingerprint
```

*Note:* if multiple keys are listed by the `gpg --list-secret-keys` command, make sure you are taking the fingerprint from the key you actively use.

After having retrieved your key's fingerprint, you can now proceed with the key upload, using either *sett* or *GnuPG*, as shown below.

**sett command:**

```
# General upload syntax:
sett verify-keylengths-and-upload-keys <key fingerprint>

# Example:
sett verify-keylengths-and-upload-keys AD1008B5F0F876630B9AF245AA2BF155EF144550
```

#### GnuPG command:

```
# General syntax:
gpg --keyserver https://keys.openpgp.org --send-key <key fingerprint>

# Example:
gpg --keyserver https://keys.openpgp.org --send-key   
AD1008B5F0F876630B9AF245AA2BF155EF144550
```

### 4.2.3 Upload your public PGP key using a web browser

1. Export your **public PGP key** to a file using the command line given below in your shell/terminal/command prompt (replace email@example.com with the email or fingerprint of your key). This should work on all platforms (Linux, Mac, Windows).

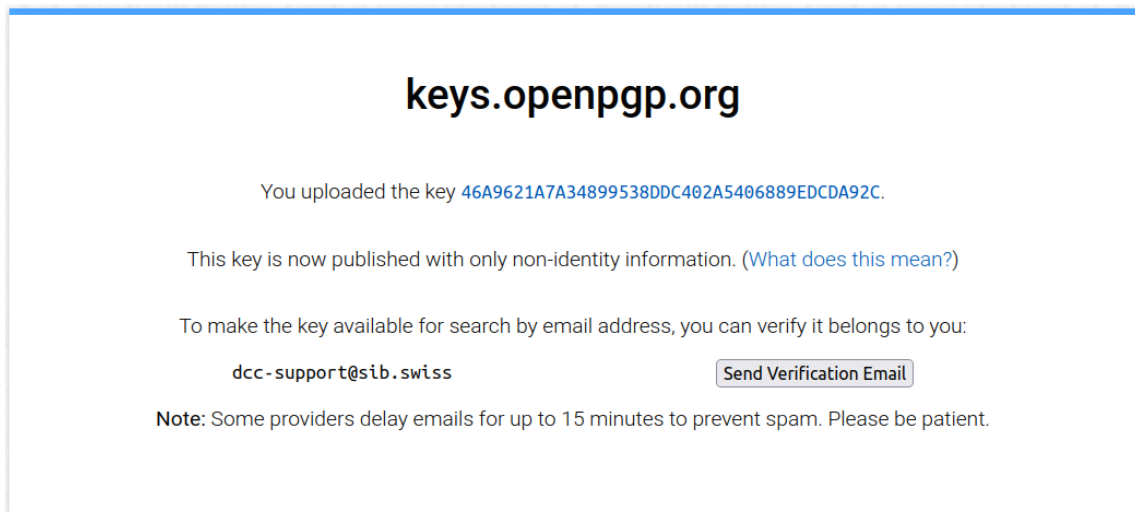
**Important:** make sure that a single key gets exported - e.g. if your email is associated with multiple keys, you should use the key's fingerprint instead of the email in the export command below.

```
# NOTE: the order of arguments matters, the '--output' option must be passed   
first!
gpg --output public_key.pgp --export --armor email@example.com

# Alternative if you have your email associated to multiple keys:
# 1. Retrieve your key's fingerprint:
gpg --list-secret-keys # For recent GnuPG versions [>= 2.2.   
x].
gpg --list-secret-keys --with-fingerprint # For older GnuPG versions.
# 2. Export your key using its fingerprint as identifier.
gpg --output public_key.pgp --export --armor   
AD1008B5F0F876630B9AF245AA2BF155EF144550
```

2. In your web browser, go to the keyserver's [upload page](https://keys.openpgp.org). Click on **Browse...** and select the file you just exported (e.g. public\_key.pgp).

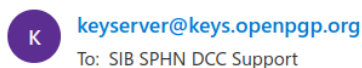
3. Click on **Upload** (blue button to the right of the text field): your key will be uploaded to the keyserver, and you should see the following confirmation message displayed:



4. The next step is to **verify your key** with the keyserver by clicking on **Send Verification Email**. The keyserver will check that you have access to the email associated with the PGP key you just uploaded.

After a key is verified, its fingerprint and email are published on the keyserver, and the key is then searchable by email (non-verified keys are only searchable by their full fingerprint).

5. **Check your email inbox:** you should have received an automated email from [keyserver@keys.openpgp.org](mailto:keyserver@keys.openpgp.org) with a **link to confirm your ownership of the key**:



Hi,

This is an automated message from keys.openpgp.org. If you didn't request this message, please ignore it.

OpenPGP key: 46A9621A7A34899538DDC402A5406889EDCDA92C

To let others find this key from your email address "dcc-support@sib.swiss", please click the link below:

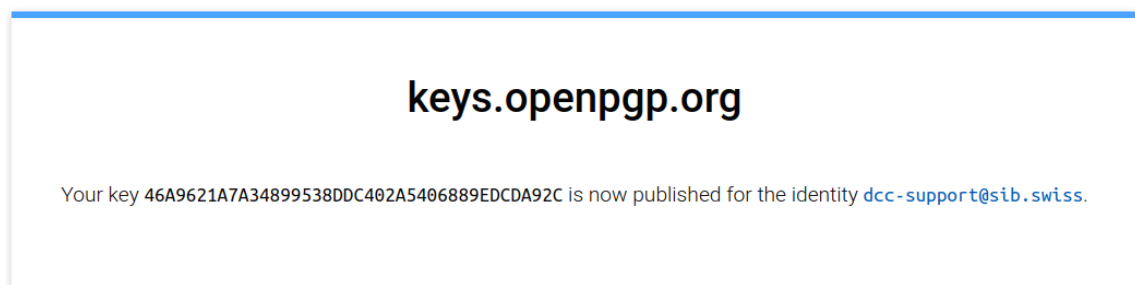
<https://keys.openpgp.org/verify/r29bozk643QmEeSmVOiQbdz9pm5MIYTAYFwweD1hzwW>

You can find more info at [keys.openpgp.org/about](https://keys.openpgp.org/about).

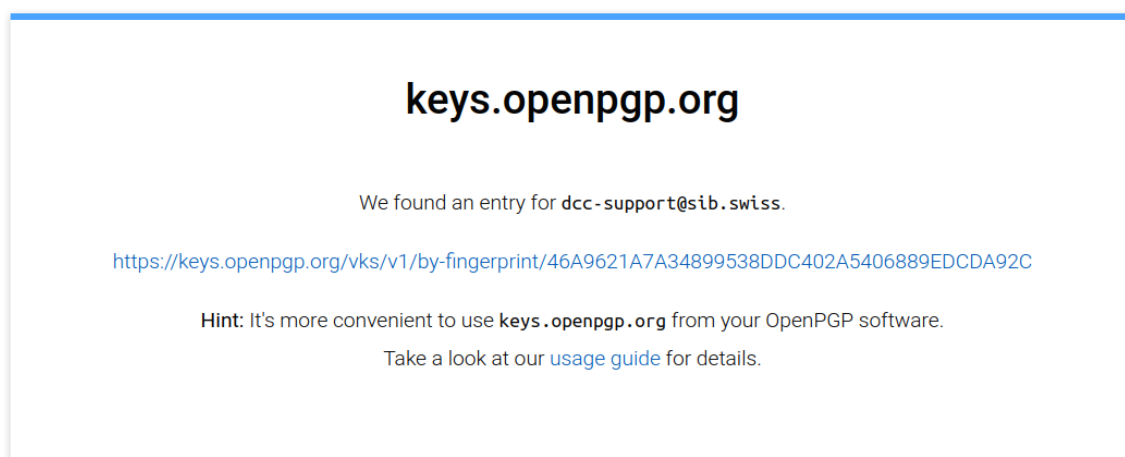
<https://keys.openpgp.org>

distributing OpenPGP keys since 2019

6. **Click on the provided link** - your key is now verified.



7. To make sure that your key was verified properly, you can go to the [keyserver home page](#) and search for the email associated with your key. You should get a message saying that a key was found for your email address, as shown in the example below:



8. You can now delete the file containing the exported public key on your local machine (e.g. `public_key.pgp`).

## 4.3 Register your public PGP key in the BioMedIT portal

---

### BioMedIT

**Registering your public PGP key in the BioMedIT portal is mandatory** to be able to use it within the BioMedIT project (e.g. create data transfer requests, encrypt data, etc.).

---

If you are *not* a BioMedIT user, this section is not relevant for you and can be skipped.

### 4.3.1 PGP key status

PGP keys used to encrypt, sign and decrypt data transiting via the BioMedIT network require the approval of the **BioMedIT key validation authority**. The information of whether a key is trusted or not is stored as **key status** in the [BioMedIT portal](#). This is the reason why all PGP keys used within BioMedIT must be registered with the BioMedIT portal.

When a PGP key is first registered in the BioMedIT portal, its status is initially set to **PENDING** (i.e. it is awaiting approval). A key must be in **APPROVED** status before it can be used to encrypt or sign data packages transiting via the BioMedIT network.

The list of key statuses is as follows:

- **PENDING**: a key approval request was submitted, but the key has not been approved yet. This is a manual process and can take from a few hours or up couple of days.
- **APPROVED**: key is approved for usage within the BioMedIT network. Only approved keys can be used to encrypt, sign and decrypt data packages that are transiting via the BioMedIT network.
- **APPROVAL-REVOKED**: approval of the key has been revoked by the BioMedIT key validation authority.
- **KEY-REVOKED**: key has been revoked by its owner.
- **REJECTED**: key is not trusted by the BioMedIT key validation authority.
- **DELETED**: key has been removed from the keyserver by its owner.
- **UNKNOWN KEY**: key has not been registered on the BioMedIT portal. If it is your own key, please register it. If it is the key of someone else, please ask them to register their key.

To verify that a key is trusted, *sett* connects to the BioMedIT portal and retrieves the status of the key. For this reason, it is important that BioMedIT users **register their PGP key with the BioMedIT portal**.

In cases where *sett* is used outside of the BioMedIT project, or the portal is not reachable, *sett* can still be used to encrypt, decrypt, and transfer data. In this case, you need to uncheck [ ] **Verify DTR** and [ ] **Verify key approval** in the **Settings** tab.

The status of a public key in the BioMedIT portal can be easily checked in the *sett-gui* application by going to the **Keys** tab, and selecting the key in the **Public key** list.

### 4.3.2 Register your public PGP key

---

**Important:** Each BioMedIT user can only have **1 active PGP key registered** in the [BioMedIT portal](#) at any time.

If you wish to replace your currently active key with another one, please connect to the [BioMedIT portal](#), go to the **“My Keys”** tab and click on the **“Retire Key”** icon (Actions column) associated with your key.

---

**Important:** The **email** associated with your PGP key **must be the same as the email that you are using with the BioMedIT portal**.

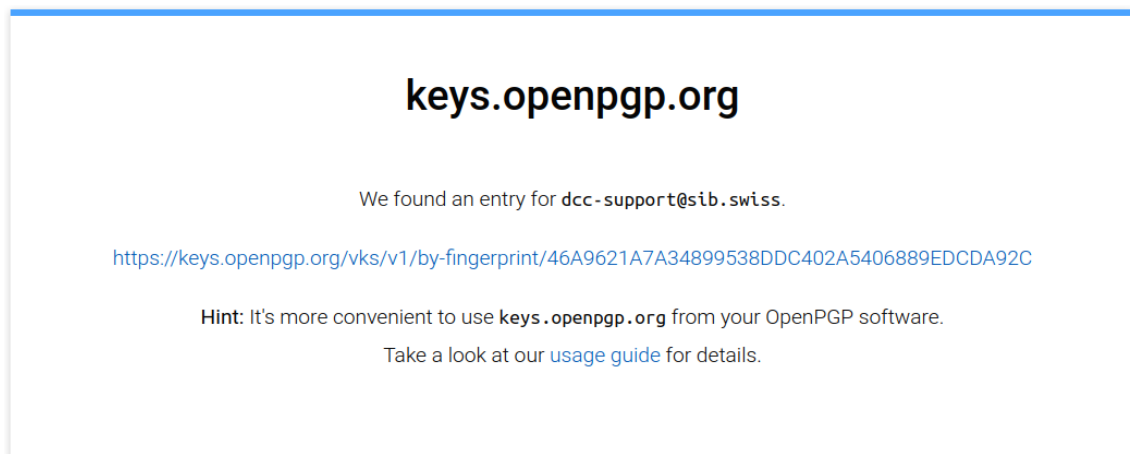
If the email associated to your PGP key is different (e.g. a shared service email is used for your PGP key), please contact the BioMedIT support.

---

To register a new PGP key with the BioMedIT portal, proceed as follows:

1. **Make sure you have successfully uploaded your key to the keyserver** and that you have **completed the email verification** procedure with the keyserver (i.e. your key must be *verified* with the keyserver).
2. To make sure that your key is present on the keyserver and is verified, go to the [keyserver home page](#) in your browser and **search for the email associated with your key**. You should get a message saying that a key was found for your email address, as shown in the example below.

Your key fingerprint will also be shown - see the right-most part of the link in this screenshot:



3. **Copy the fingerprint** (40-character string) of your key.
4. **Log in** to the [BioMedIT portal](#).
5. Go to the **My Keys** tab.
6. Click on the **“+ KEY”** button, a dialog box will open.  
*Note:* if the button is missing, it is probably because you already have an active key in the portal. Each user can only have 1 active key at a time - see the information box above.
7. Enter your full key fingerprint (must be exactly 40 characters long) in the dialog box, then press the green search icon to the right.

This will retrieve the **user ID** and **email address** associated with the fingerprint from the keyserver and display them in the dialog box.

8. **Verify the user ID and mail address.** If they are correct for your PGP key, then click on **Confirm**.
9. A request to approve your key has now been sent to the BioMedIT key validation authority. Generally requests are processed quickly (in a matter of hours), but occasionally it might take slightly longer as this is a manual process.

Please contact the BioMedIT support if your key has not been approved after a couple of days.

## 4.4 Download public PGP keys from the default keyserver

In order to encrypt data for a specific recipient (who will be able to decrypt it), you will need to have the public PGP key of that recipient(s) available in your local keyring. *sett* allows user to easily search and retrieve public PGP keys from the keyserver specified in the *configuration file*.

---

### BioMedIT

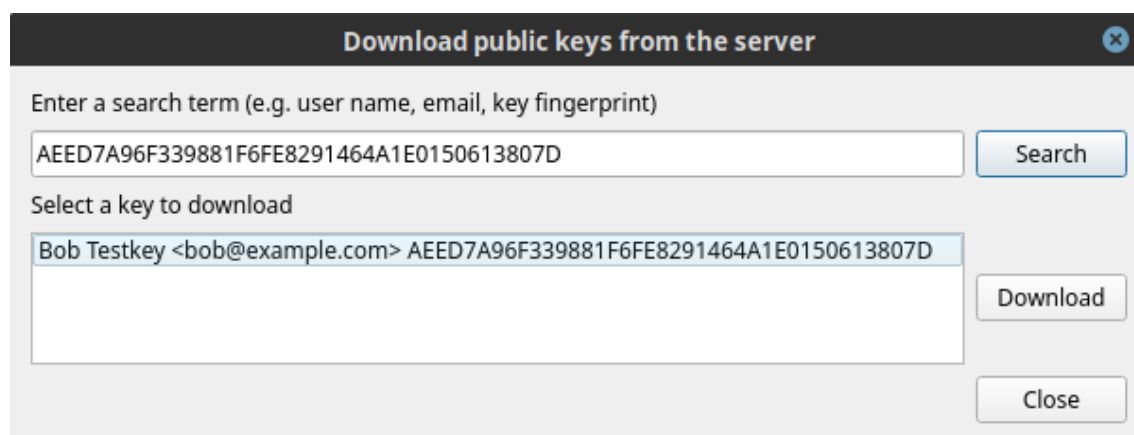
By default, *sett* is configured to search, upload and download keys from the public keyserver at <https://keys.openpgp.org>. If your configuration still points to the former keyserver (keyserver.dcc.sib.swiss), please change it to <https://keys.openpgp.org>.

---

### 4.4.1 Download PGP keys with sett-gui

To download a public PGP key from the keyserver:

1. In the **Keys** tab, click on **Download keys**. A dialog box will open, allowing to search for public keys stored on the keyserver.
2. In the search field, enter either the user email address or fingerprint of the public key you are looking for and hit **Search**: the matching keys found on the keyserver will be displayed.



For instance, if searching for the key “Bob Testkey <bob@example.com>” with fingerprint “AEED7A96F339881F6FE8291464A1E0150613807D”, one can search for either “bob@example.com” or “AEED7A96F339881F6FE8291464A1E0150613807D”.

3. When you are confident that the key is genuine, select it and click on **Download**. You should now see your recipient’s key listed in the **Public keys** box.
4. Select the newly downloaded key in the **Public keys** list and verify that it is marked as “**This key is approved**”, printed in green.

**Important:** if the message is red (“This key is not approved”), the **status of the key** (indicated at the end of the line) will give more details. Please see the *PGP key status* section for details.

### 4.4.2 Download PGP keys in command line

The following commands show how to download a public PGP key from the OpenPGP keyserver in command line. To download keys from a different keyserver, simply replace the URL of the keyserver.

```
# Search and download key from OpenPGP keyserver.
# The <search term> can be either email or fingerprint.
gpg --keyserver https://keys.openpgp.org --search-keys <search term>

# Example: download the public key of the SPHN Data Coordination Centre.
gpg --keyserver https://keys.openpgp.org --search-keys B37CE2A101EBFA70941DF885881685B5EE0FCBD3
```

Alternatively, if the fingerprint of the key to retrieve is known the `--recv-key` option can be used to directly download the key without first searching for it:

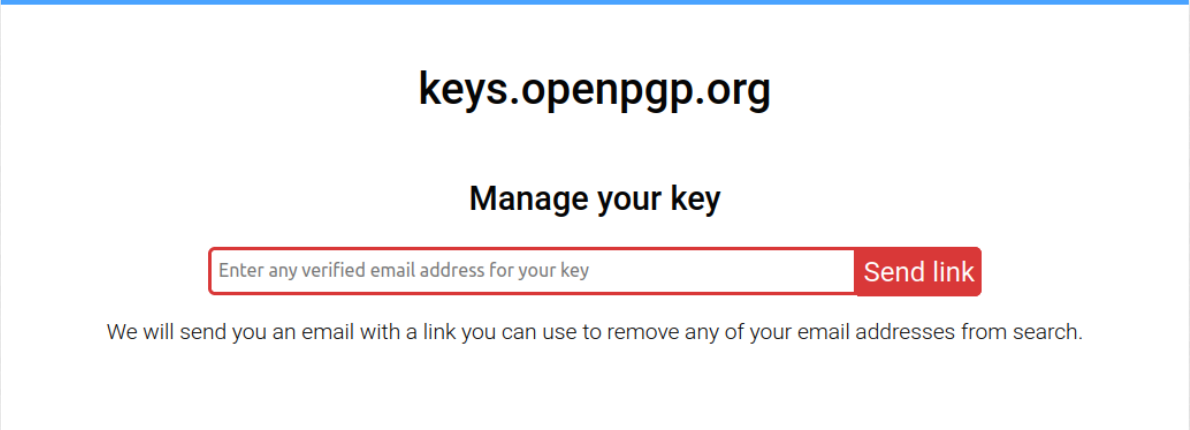
```
# Example: download the public key of the SPHN Data Coordination Centre.
gpg --keyserver https://keys.openpgp.org --recv-key B37CE2A101EBFA70941DF885881685B5EE0FCBD3
```

**Note:** If you are behind a proxy, you will need to extend the `gpg` command with the following *http-proxy* option:  
`--keyserver-options http-proxy=http://proxy:port`

### 4.4.3 Remove your public PGP keys from the keys.openpgp.org keyserver

While it is not possible to remove an actual key from the keyserver, it is possible to remove all personal identification from it (user ID and email). Such keys are called **unverified**.

To remove personal information associated with a key, go to the keyserver's [manage key page](#), enter the email associated to the key and click on **Send link**.



You will receive an email with further instructions on how to proceed to remove your key's user ID and email from the keyserver.

**Warning:** Removing a key's user ID and email from the keyserver makes it no longer searchable by email. It remains searchable by its full fingerprint.

**Important:** keys without user ID and email cannot be used within the BioMedIT network. Only remove your personal identifying information from a key if you are no longer using it.



## 4.5 Delete PGP keys from your local machine

Deleting public and/or secret PGP keys from your local certificate store **is not possible with `_sett_`**. This is intentional, and users are encouraged to **revoke their key** instead.

Should you nevertheless wish to delete a key from your local certificate store (e.g. because you created a test key that is no longer needed), it is possible to do so by manually deleting the files containing the certificates using your file explorer or a shell command.

In the local *sett* certificate store, certificates are stored as files named after the fingerprint of the certificate's primary key. For instance, a certificate with fingerprint `3b17f529665fe012ef54f4a1714fdf98b6e828df` would be stored under:

- Public certificate: `<certstore-path>/pgp.cert.d/3b/17f529665fe012ef54f4a1714fdf98b6e828df`
- Secret certificate: `<certstore-path>/pgp.cert.d.sec/3b/17f529665fe012ef54f4a1714fdf98b6e828df`

The location of the certificate store (`<certstore-path>` above) is operating-system dependent:

- **Linux:** `~/.local/share/pgp.cert.d` for public certificates, and `~/.local/share/pgp.cert.d.sec` for secret certificates.
- **Windows:** `%UserProfile%\AppData\Roaming\pgp.cert.d` for public certificates, and `%UserProfile%\AppData\Roaming\pgp.cert.d.sec` for secret certificates.
- **MacOS:** `~/Library/Application Support/pgp.cert.d` for public certificates, and `~/Library/Application Support/pgp.cert.d.sec` for secret certificates.

## 4.6 Revoke your PGP key

If a private PGP key has been compromised, is no longer usable (e.g. password is lost), or should no longer be used for any other reason, it must be revoked.

A prerequisite for revoking a PGP key is to have generated a *revocation certificate* for it. If the PGP key to revoke was generated with *sett*, you should in principle already have a revocation certificate ready to use. If you do not have a revocation certificate yet, please generate one by referring to the *Generate a new key pair in command line* section.

**Warning:** A revoked key cannot be “un-revoked”. Only revoke a key if you are certain you want to do so. Revoked keys can no longer be used to encrypt/decrypt data with *sett*.

### 4.6.1 Revoke your PGP key with *sett-gui*

To revoke a public key in *sett-gui*:

1. In the **Keys** tab, click on **Import key**.
2. Select the *revocation certificate* for the public key you want to revoke. This will import the revocation certificate and immediately revoke the key - there is no confirmation requested.

**Warning:** proceed with caution, a revoked key cannot be “un-revoked”.

3. When selecting the revoked key from the **Public keys** list, you should now see a red “**key has been revoked**” warning listed under the key's signatures. From this point on, the key can no longer be used with *sett*.



<b>User ID</b>	Alice Smith University of Basel <alice.smith@example.com>
<b>Key ID</b>	867A6FCF40E643D4
<b>Key fingerprint</b>	C388F057FB1BD5753F51D24F867A6FCF40E643D4
<b>Key length</b>	4096
<b>Signatures</b>	
Alice Smith University of Basel key has been revoked	

4. If you have previously shared your key via a keyserver (e.g. [keys.openpgp.org](https://keys.openpgp.org)), you must also re-upload your revoked key to that keyserver.

This will allow other users to update their local copy of your public key, informing them that it is no longer valid. To upload your revoked public key, select it from the **Public keys** list, then click on **Upload keys** to upload it to the keyserver.

If your key was never present on any keyserver, this step should be skipped.

## 4.6.2 Revoke your PGP key in command line

Key revocation is not supported in the *sett* command line and must be done directly with *gpg*, as shown in the following example:

```
# Example:
gpg --import revocation_certificate.asc
```

After a key has been revoked, it must be uploaded again to any keyserver(s) where it is present, so that the revocation can be shared with others. This can be done with *sett* as illustrated in the example below:

```
# Example: a key with fingerprint AD1008B5F0F876630B9AF245AA2BF155EF144550
# was revoked and must now be uploaded to the keyserver.
sett verify-keylengths-and-upload-keys AD1008B5F0F876630B9AF245AA2BF155EF144550
```

## 4.7 Migrating keys from the GnuPG keyring to the sett certificate store

Starting from version 4.4.0, *sett* no longer uses *GnuPG* as default encryption backend. Instead it uses an embedded library - *Sequoia* - and its own keystore (reverting to using *GnuPG* is however possible by enabling the *legacy\_mode* option in the *configuration file* or “Use *GnuPG* legacy mode” in the settings tab of *sett-gui*).

As a result, Open PGP keys located in your *GnuPG* keyring are no longer automatically detected by *sett*, and they must be migrated to the new *sett* certificate store.

If you are a user with PGP keys in your *GnuPG* keyring that you are using with *sett*, then these keys must be migrated to the new *sett* certificate store. Using *sett-gui* (the GUI version of *sett*), this procedure is straightforward:

1. In *sett* GUI, go to the **Keys** tab.
2. Click on the **Migrate secret key from GnuPG** icon (pictogram of a square duplicating itself).
3. A pop-up should open, allowing you to **select the secret PGP key** to migrate.
4. **Enter the password** associated with your secret PGP key, then click on **Migrate key**.
5. Done. Your secret PGP key has now been imported into *sett*’s certificate store and you should see it listed under in the **Private keys** and **Public keys** boxes.

6. To migrate public keys (i.e. keys from other people), please simply re-download them from the keyserver by clicking on **Download keys from keyserver** (pictogram of a cloud with arrow pointing down).

Note: migrating keys from the GnuPG keyring to the *sett* certificate store is currently not possible in command line mode (sett CLI).

The location of the *sett* certificate store is the following:

- **Linux:** `~/.local/share/pgp.cert.d` for public certificates, and `~/.local/share/pgp.cert.d.sec` for secret certificates.
- **Windows:** `%UserProfile%\AppData\Roaming\pgp.cert.d` for public certificates, and `%UserProfile%\AppData\Roaming\pgp.cert.d.sec` for secret certificates.
- **MacOS:** `~/Library/Application Support/pgp.cert.d` for public certificates, and `~/Library/Application Support/pgp.cert.d.sec` for secret certificates.

## 4.8 Introduction to public-key cryptography, PGP and GnuPG

Public-key cryptography is a method for secure communication between two or more users. In this system, each user has a pair of unique keys consisting of a **private key** and a **public key**. Public and private keys are linked in the sense that, data encrypted with a given public key can only be decrypted with the matching private key, and data signed with a given private key will only be recognized by the matching public key.

Because these keys are based on the [OpenPGP](#) protocol, they will here be referred to as **PGP keys**.

Public and private PGP keys:

- **Public keys** are used to encrypt data, as well as for verifying signatures made on files or emails. By design, public keys are intended to be shared with other people and therefore no particular effort is required to keep them secret. In fact, public keys are often uploaded to public servers, known as **keyservers**, where they are accessible to anyone. No password is required to use a public key.

Typically, public keys are used by data senders to encrypt data for one or more recipient(s), and by data recipients to verify signatures of files or emails (to ensure the sender is genuine).

- **Private keys**, sometimes also referred to as a **secret keys**, are used to decrypt data, sign files and sign other people's public keys. To increase security, private keys should always be password protected.

---

**Important: Private keys and their password must be kept secret at all times. Never share your private key or password with anyone.**

Private keys should be stored in a directory only accessible by the key owner, and their password should be stored in a password manager.

---

Typically, private keys are used by data recipients to decrypt data, and by data senders to sign the files they encrypt.

---

**Important:** If anyone else than the legitimate owner has access to a private key and/or its password, **the key pair is considered compromised**. It *must be revoked*, and never used again.

---

### **Warning: GPG security warning**

The GnuPG software keeps key passwords entered by the users temporarily stored for the duration of a user's session - i.e. until the user logs out of a session (sometimes the duration of the temporary storage is also limited in time). This is done for convenience reasons, so that users do not always have to re-enter their password.

A security implication is that if someone has physical access to your computer, that person might now be able to use your private key without having to know its password.

*sett* uses the open source implementation of public-key cryptography provided by the GNU Privacy Guard software (**GPG** or **GnuPG**, in short). GnuPG is a command line tool developed by the Free Software Foundation that uses the **PGP OpenPGP** implementation to encrypt and decrypt files.

A detailed documentation of PGP and GnuPG can be found in this online [manual](#).

It is also possible - and often desirable - to both encrypt and sign a file. This ensures that the data can only be read by the intended recipient, and that the recipient can be confident the sender is legitimate. This is precisely what *sett* does:

- **Encrypting files**, so that only the intended recipient(s) can read them.
- **Signing files**, so that the recipient(s) can trust the sender is genuine.

### 4.8.1 Key fingerprints

Each pair of public/private PGP keys is identified by a unique **fingerprint**. Fingerprints are 40 characters long hexadecimal strings (digits and upper case A-F letters) that look like this:

```
238565936FCFF3F200219990941A3EC20555F781
```

Since nothing is preventing two PGP keys to have the same user name and email address, it is critical that users **always verify the genuineness of new keys** before (or just after) importing them into their local keyring (i.e. their local PGP key database).

Ensuring a key is genuine can be done in two different ways:

- Ask the key owner to **provide their key's fingerprint via a trusted communication channel** (e.g. over the phone), and then verify that the fingerprint of the newly imported key does indeed match the fingerprint communicated to you by its owner.
- Using *sett-gui*, verify that the *key status* of the key is **APPROVED** (can only be checked after you imported the key).

### 4.8.2 File encryption

In public key cryptography, the sender **encrypts** a file using one or more recipient(s) public key(s). Once a file is encrypted, no one can read the file without having access to a private key that matches the public key(s) used for encryption. This ensures that only the intended recipient(s) can decrypt the file, because they are the only one to have access to the matching private key.

### 4.8.3 File signing

The objective of **file signing** is to guarantee to the recipient of a file (or email) that the sender is genuine, and not someone else trying to impersonate the sender.

To achieve this, the sender signs the file with their private key (password required), and shares their public key with the recipient (typically via a keyserver). The recipient can then validate the authenticity of the signature using the public key of the sender. Since public keys are non-sensitive, they can be distributed publicly. In fact they are intended for this purpose, hence their name.

## 4.8.4 Revocation certificates

In the unfortunate event that a user either i) forgets their private key's password or ii) have their private key and password stolen/compromised, they will need a way to let other people know that their public key should no longer be trusted and used.

This is because:

- If the password was forgotten: the key owner won't be able to decrypt data anymore.
- If the private key was compromised: someone else might be able to decrypt data encrypted with the public key, and to illegitimately sign files!

This situation is what **revocation certificates** are for: by applying a revocation certificate to a public key, and then sharing the revoked key with others (e.g. via a keyserver), the key owner signals that their key is now "revoked" and should no longer be trusted nor used. After a key has been revoked, it can no longer be used to encrypt/decrypt data with *sett*.

Revocation certificates can be generated at anytime from the private key, but the best practice is to generate them directly after a new key pair is created. This ensures that the revocation certificate will be available even if the private key or its password is lost.

---

**Note:** *sett-gui* will automatically generate a revocation certificate each time a new key is created.

---

Since anyone with access to a revocation certificate will be able to revoke the associated key, revocation certificates must be stored securely - e.g. in a password manager - and should never be shared with anyone.

---

### Important:

- Revocation certificates **must be stored in a safe location** (e.g. a password manager) and **never shared with anyone**.
  - It is best to generate a revocation certificate **immediately after** a new public/private key pair is created.
  - The actual procedure to revoke a key is detailed in the [Revoke your PGP key](#) section.
- 

## 4.8.5 Exchanging public keys via a keyserver

Encrypting files for a specific recipient requires to have the recipient's public key in one's local **keyring** (a keyring is a local database containing PGP keys). Similarly, verifying a signature on a file or a public key requires to have the signee's public key available in one's local keyring.

Public keys are not sensitive data, and therefore can in principle be sent unencrypted via email. However, when having frequent key exchanges between multiple actors, sending public PGP keys around by email quickly becomes cumbersome. A solution to this problem is using a so called **keyserver** to share public keys. Keyservers are public or private servers whose sole purpose is to store public PGP keys and allow users to search for them. In addition, public keyservers often form a network and are regularly synchronized among themselves to ensure redundancy. Private keyservers have the same role, but do not share/synchronize the stored public keys with any other server, and possibly have a restricted access policy so that only authorized people can search for keys.

---

### BioMedIT

Within the BioMedIT project, **all exchanges of public keys are done via the public keyserver at <https://keys.openpgp.org>**. This keyserver can be accessed directly from the *sett* application (recommended), via a web browser, or through command line programs such as *gpg*.

**New:** they keyserver used to exchange public PGP keys within the BioMedIT project has been changed in September 2022. If you have used *sett-gui* before, make sure you have set the *Keyserver URL* in the *Settings* tab set to

<https://keys.openpgp.org>. Updating to the latest version of *sett* is also required and will modify this setting automatically for you.

---

## 4.9 Troubleshoot the “Network is unreachable” and “No keyserver available” error

On Mac OS Big Sur and possibly above, the `gpg` command to upload or download keys might end up in a `Network is unreachable` error.

To solve this issue, try the following commands:

```
echo "standard-resolver" >> ~/.gnupg/dirmngr.conf  
gpgconf --reload dirmngr
```

Since `gpg` version 2.1, a daemon called `dirmngr` takes care of accessing the OpenPGP servers. The configuration above forces `dirmngr` to use the system’s standard DNS resolver code. The second command restarts the `dirmngr` daemon, which will then pick up the new configuration.



## GENERATING SSH KEYS

SSH (Secure SHell) keys are pairs of small text files that are used to securely identify users and give them access to remote servers, e.g. when transferring data via SFTP.

SSH keys use [public-key encryption](#) and always come in pairs: a public and a private (or secret) key.

- **Public key:** the public key of an SSH key pair is meant to be placed on the remote machine to which a user wants to connect. Public keys are non-sensitive, and are typically shared by users with system administrators, who will place them on the machine to which the user is granted access.
- **Private key:** the private key of an SSH key pair is what uniquely identifies a user as the legitimate owner of a public key. In other words, having the private key that matches a given public key will give access to any machine on which a copy of public key is stored. Private SSH keys are sensitive information: they **must be kept private at all times** and should **never be shared with anyone** - not even your system administrator. Private keys can (and should) be protected by a password, so that even if someone else has access to them, they remain unusable.

Generating a new pair of SSH keys must be done only once, and, in the context of *sett*, is only needed if you intend to transfer data. If you are a user who only decrypts data, you do not need an SSH key.

Also, do not confuse SSH keys - used to identify yourself on a remote server - *with PGP keys* - used to encrypt and sign data.

To generate a new SSH key pair, type the command below in your terminal (Linux and Mac) or PowerShell (Windows users - to start it, search for “powershell” in the Start menu). Note that you must replace “alice@example.org” with your own email. This will generate an SSH key pair using the “ed25519” algorithm, currently the most secure public-key algorithm:

```
ssh-keygen -a 100 -t ed25519 -C "alice@example.org"
```

**Windows users** who do not have the *ssh-keygen* command installed, please see [Windows users: enabling the ssh-keygen command](#) below.

When executing the *ssh-keygen* command above, you will be prompted for the following information:

1. The name and location where to save the newly created keys. Here you should simply accept the default values by not entering anything and pressing “Enter” on your keyboard. Default locations are `~/.ssh/id_ed25519` on Linux and MacOS, and `C:\Users\%username%\ .ssh\id_ed25519` on Windows.
2. A password to protect your private SSH key against illegitimate use. Please use a password that is long enough ( $\geq 12$  characters) and composed only of [ASCII characters](#).

When the command completes, two new files are produced: `id_ed25519.pub` (the public key) and `id_ed25519` (the private key).

**Warning:** Remember that the file containing the private key (`id_ed25519`) must be kept secret. **Never share it with anyone**, not even your system administrator.

On Linux and MacOS systems, after the public key is generated, its permissions must be changed with the following command (this step is not needed for Windows users):

```
chmod 600 ~/.ssh/id_ed25519.pub
```

## 5.1 Windows users: enabling the ssh-keygen command

Not all versions of windows come with the *ssh-keygen* command pre-installed. If this command is unavailable on your machine, please install it as follows:

1. Open the windows settings (shortcut: windows key + i).
2. Search for, and select, “Add an optional feature”.
3. Click on “Add a feature”.
4. Search for, and select, “Open SSH Client”.
5. Click “Install”.
6. Restart your computer.



## ENCRYPTING, TRANSFERRING AND DECRYPTING DATA WITH SETT

### 6.1 Encrypting files

*sett* allows the encryption of any combination of individual files and directories.

The files are first compressed into a single `data.tar.gz` archive, which is then encrypted with the public key of one or more recipient(s), and signed with the sender's key. The encrypted data (`data.tar.gz.gpg`) is then bundled with a **metadata file** - a plain text file that contains information about who is sending the file and to whom it should be delivered - into a single `.zip` file. The specifications of the output `.zip` files produced by *sett* are described in the *sett packaging specifications* section.

The following OpenPGP certificates **must be present in the local sett certificate store** in order to perform data encryption:

- The secret certificate of the data sender (i.e. the person encrypting the data). This is needed for signing the data.
- The public certificate of all data recipients (people for whom the data is being encrypted). *sett* supports multi-recipient data encryption, allowing the encrypted file to be decrypted by multiple recipients.

*sett* also ensures the integrity of the transferred files by computing checksums on each file that is packaged, and adding this information to the encrypted data. The integrity of each file is verified automatically upon decryption of the file by *sett*, providing the guarantee that all files were transferred flawlessly.

---

#### BioMedIT

**Data Transfer Requests:** each data transfer into the BioMedIT network must have an authorized **Data Transfer Request ID (DTR ID)**. This ID **must be specified at the time the data is encrypted** (see below). The ID is added to the encrypted file's metadata information by *sett*. A valid and authorized DTR ID value is mandatory for any data transfer into the BioMedIT network. Non-compliant packages will be rejected.

**Recipients:** each data transfer into the BioMedIT network must be to a recipient assigned to the role of Data Manager for the given project. The recipient's PGP key must also be approved by the BioMedIT key validation authority. If these conditions are not met, *sett* will not encrypt the data.

---

### 6.1.1 Output file naming scheme

By default, encrypted output files produced by *sett* are named after the pattern:

`<project code>_<YYYYMMDD>T<HHMMSS>_<optional suffix>.zip`

where:

- `<project code>` is the abbreviation/code associated with the project. If no **DTR ID** value was provided or if **Verify DTR** is disabled, no project code is added as a prefix to the output file name.
- `<YYYYMMDD>` is the current date (Year, Month, Day).
- `<HHMMSS>` is the current time (Hours, Minutes, Seconds).
- `<optional suffix>` is an optional, custom text that can be added to the file name.

Example: `demo_20220211T143311_sib.zip`, here `demo` is the project code and `sib` is an optional suffix.

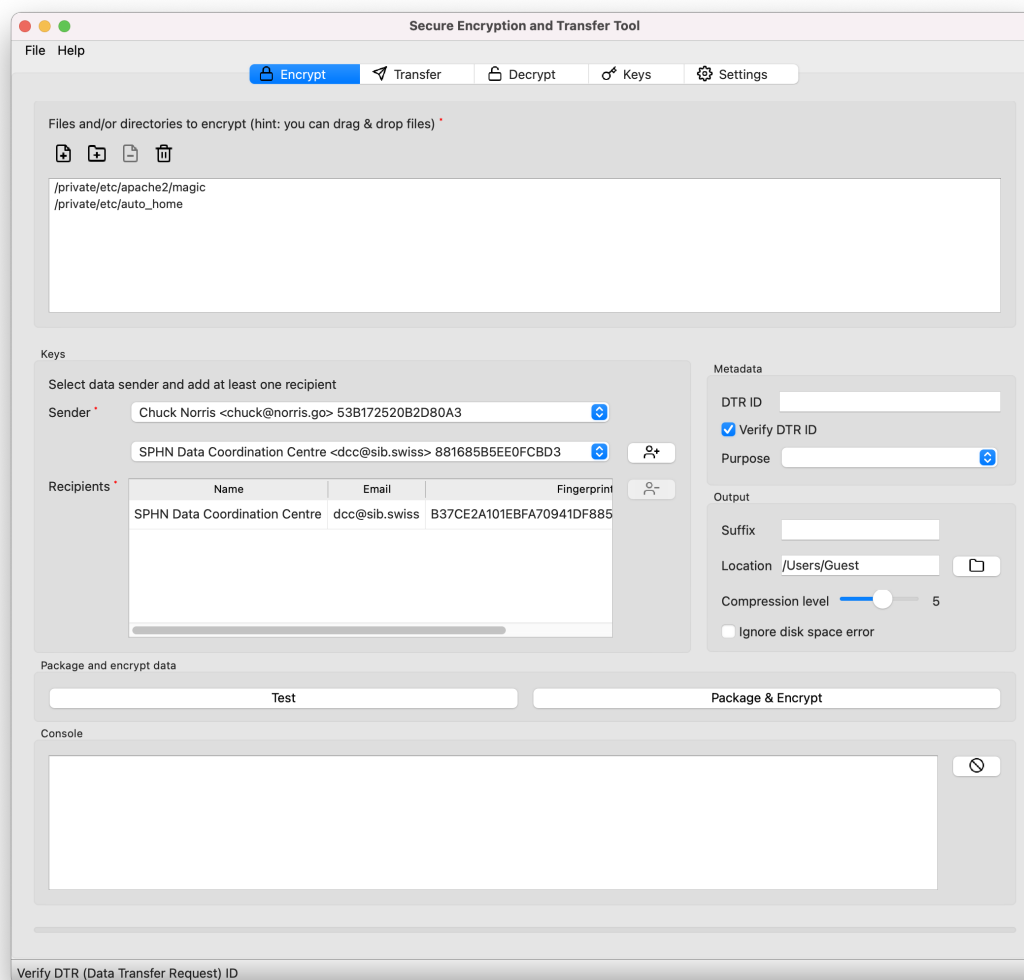
The value for the optional suffix can be permanently set in the *Settings* tab of the *sett-gui*, or in the [sett configuration file](#).

Using the *sett* command line, it is possible to completely override the above output file naming scheme by passing the `--output` option. Overriding the naming scheme is not possible when using *sett-gui*.

### 6.1.2 Encrypting data with sett-gui

To encrypt data:

1. Go to the **Encrypt** tab of the *sett* application.



2. **Select files and/or directories to encrypt:** using the **Add files** and **Add directory** buttons, select at least one file or directory to encrypt.

After adding files/directories, they will be listed in the top box of the tab (see figure above).

3. **Select data sender:** in the drop-down list found under **Sender**, select your own PGP key (you are the data sender). For most users, there should in principle be only one key in the **Sender** drop-down menu: their own key.

---

**Note:** The **Sender** key is used to sign the encrypted data, so that the recipient(s) can be confident that the data they receive is genuine.

---

4. **Select data recipients:** add one or more recipients by selecting them from the drop-down list found under **Recipients** and clicking the **+** button. Recipients are the people for whom data should be encrypted: their public PGP key will be used to encrypt the data, and only they will be able to decrypt it.

---

### BioMedIT

Only recipients assigned to the role of **Data Manager** of the project for which data is being encrypted are permitted as data recipients.

---

5. **DTR ID:** Data Transfer Request ID associated to the data package that is being encrypted. Specifying a **valid DTR ID** is mandatory to transfer data into the BioMedIT network.

For data not intended to be transferred into the BioMedIT network, the **DTR ID** field can be left empty (or set to any arbitrary value). In this case, **Verify DTR** must be disabled (in the **Settings** tab).

---

**BioMedIT**

**DTR ID** field is mandatory. Only files encrypted with a valid and authorized **DTR ID** value can be transferred into the secure BioMedIT network. For this reason, BioMedIT users should always leave the **Verify DTR** checkbox enabled.

---

6. **Purpose:** purpose of the data transfer, please select either PRODUCTION or TEST, or leave it empty.

---

**BioMedIT**

This field is mandatory.

---

7. **Output suffix (optional):** optional suffix value to be appended at the end of the file name. A `_` separator is automatically added and does not need to be part of the suffix.
- Only regular alphanumeric, `-` and `_` characters are allowed in the output suffix.
  - The value for the optional suffix can be permanently set in the *Settings* tab.
  - For more details on the output file naming scheme used by *sett*, please refer to the [output file naming scheme section](#).
8. **Output location:** directory where the encrypted file should be saved.

By default, output files are saved to the user's home directory.

9. **Compression level** slider: amount of compression to apply to the input data when packaging it.

Compression values range between 0 (no compression) and 9 (highest compression). Higher compression level results in smaller encrypted output files but require more computing time.

The default compression level of 5 offers a good balance between output compression and time needed to perform the task. An illustration of compression ratio vs. time is given in the [sett benchmarks section](#).

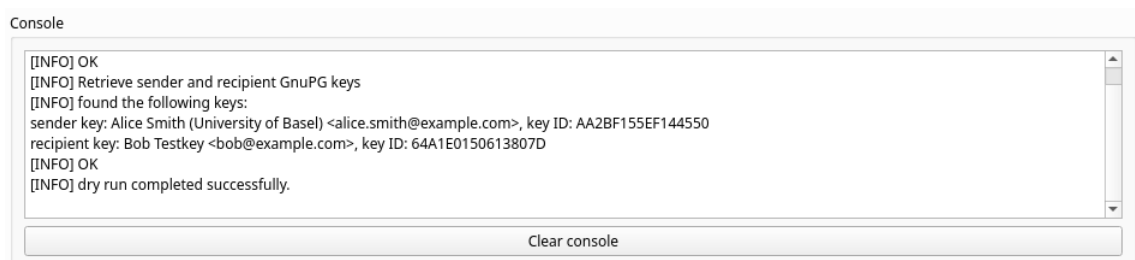
If compression is not required, e.g. because the input data is already in a compressed form, the compression level should be set to 0 in order to speed-up the packaging task. Performing compression outside of *sett* can be useful when [working with large files](#).

10. **Ignore disk space error:** disable disk space check before data encryption.

By default, *sett* verifies that there is enough free disk space available to save the output file before starting to compress and encrypt data. If this is not the case an error message is displayed and the operation is aborted. Since the compression ratio of the input data cannot be known in advance, *sett* uses the conservative estimate that the minimum disk space required is equal to the total size of all input files to be encrypted.

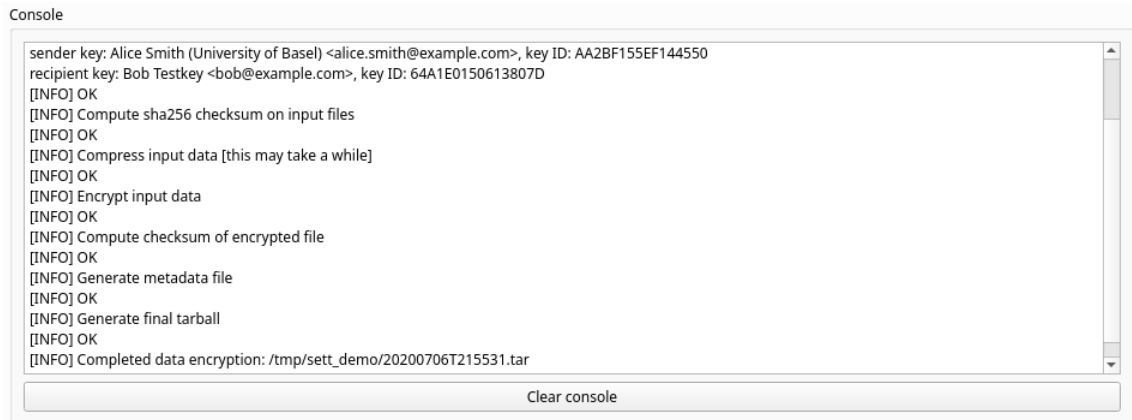
If users think this is too conservative, this verification can be disabled by turning the **Ignore disk space error** checkbox on.

11. **Encryption Test:** a test run of the data to encrypt can be performed by clicking the **Test** button. This will check that all the specified input files can be found on disk, and run additional checks if **Verify DTR** settings is enabled (default).



- You are now ready to compress and encrypt the data: click **Package & Encrypt**. A pop-up will appear, asking for the password associated with the sender's key. After the password is entered, data compression and encryption will start. Progress and error messages are displayed in the **Console** box.

When the encryption completed successfully, the **Console** should display a message that reads: "Completed data encryption" followed by the location and name of the output file, as illustrated in the example below.



At this point, all input files are compressed, encrypted and bundled into a single .zip file. Data has *not* yet been transferred to the intended recipient.

### 6.1.3 Encrypting data on the command line

The `sett` command to encrypt data is the following. Note that the `SENDER` and `RECIPIENT` values can be specified either as a PGP key **fingerprint**, or as an **email address**.

```
# General syntax:
sett encrypt --sender SENDER --recipient RECIPIENT --dtr-id DATA_TRANSFER_ID --
↳purpose PURPOSE --output OUTPUT_FILENAME_OR_DIRECTORY FILES_OR_DIRECTORIES_TO_
↳ENCRYPT

# Example:
# long command line options:
sett encrypt --sender alice@example.com --recipient bob@example.com --dtr-id 42 --
↳purpose PRODUCTION --output test_output ./test_file.txt ./test_directory
# short command line options:
sett encrypt -s alice@example.com -r bob@example.com -t 42 --purpose PRODUCTION -o
↳test_output ./test_file.txt ./test_directory --dry-run
```

Data can be encrypted for more than one recipient by repeating the flag `--recipient`, e.g. `--recipient RECIPIENT1 --recipient RECIPIENT2` option:

```
# In this example, Alice encrypts a set of files for both Bob and Chuck.
sett encrypt --sender alice@example.com --recipient bob@example.com chuck@example.com
↳FILES_OR_DIRECTORIES_TO_ENCRYPT
```

Adding the `--dry-run` option will run the `encrypt` command in test mode, i.e. checks are made but no data is encrypted.

The data compression level used by `sett` can be manually adjusted using the `--compression-level` option. Compression levels value must be integers between 0 (no compression) and 9 (highest compression). Higher compression levels produce smaller output files but require more computing time, so you may choose a lower level to speed-up compression (e.g. `--compression-level=1`), or a higher level (e.g. `--compression-level=9`) to produce smaller output files. The default level is 5.

Before encrypting data, `sett` verifies that there is enough free disk space available on the local machine to save the encrypted output file (relevant is the current working directory or target folder pointed by `--output`). If this is not

the case an error message is displayed and the operation is aborted. Since the compression ratio of the input data cannot be known in advance, *sett* uses the conservative estimate that the minimum disk space required is equal to the total size of all input files to be encrypted. If users think this is too conservative (e.g. because they know that their data compresses well), this verification can be disabled by passing the `--force` option.

To automate the encryption process, the `--passphrase-cmd` option can be used to specify an external command that returns the PGP key password to the standard output.

---

**Important:** When using `--passphrase-cmd`, make sure that the external command and the password store are **secure**.

---

*sett* performs **DTR** verification if `verify_dtr` is enabled in settings (default). For non-BioMedIT-related transfers, `verify_dtr` should be set to `false` (`--dtr-id` and `--purpose` are optional in this mode).

---

## BioMedIT

A valid DTR ID must be specified via the `--dtr-id` option and purpose via `--purpose`.

---

An optional output suffix can be added to the *sett* output files by passing the `--output-suffix` option. Alternatively, the value for the optional suffix can be permanently set in the [sett configuration file](#). For more details on the output file naming scheme used by *sett*, please refer to the [output file naming scheme section](#).

To completely override the *sett* output file naming scheme, the `--output` option can be used to specify the path and name that the output file should have.

## 6.2 Transferring files

Data packages can be transferred to remote servers that support one of the following protocols:

- SFTP
- S3 object storage
- Liquid Files

---

**Important:** Only files encrypted with *sett*, or files that follow the [sett packaging specifications](#) can be transferred using *sett*.

---

By default, *sett* verifies data packages before initializing file transfers. These checks are required within the BioMedIT network, but can be skipped in other contexts by disabling **Verify DTR**, **Verify package name**, and/or **Verify key approval** checkboxes in the application settings.

- **Verify DTR:** A valid and authorized DTR (Data Transfer Request) ID is required in the package metadata.
- **Verify package name:** Package name must match the pattern `<project_code>_<date-format>_<package_name_suffix>.zip`, where:
  - **project\_code** is the abbreviated project name of the project whose DTR ID was used during data encryption. If no DTR ID value was given at encryption time, or if checking DTR ID is disabled, the pattern against which files named are verified becomes `<date-format>_<package_name_suffix>.zip`.
  - **date\_format** is the date and time when the data package was created, as specified in the [sett packaging format](#).
  - **package\_name\_suffix** is the optional suffix appended to packages names when they are created. When using *sett-gui*, the suffix value is taken from the Encrypt tab (which itself can be taken from the config file). When using the command line, the suffix value is taken from the [sett configuration file](#).

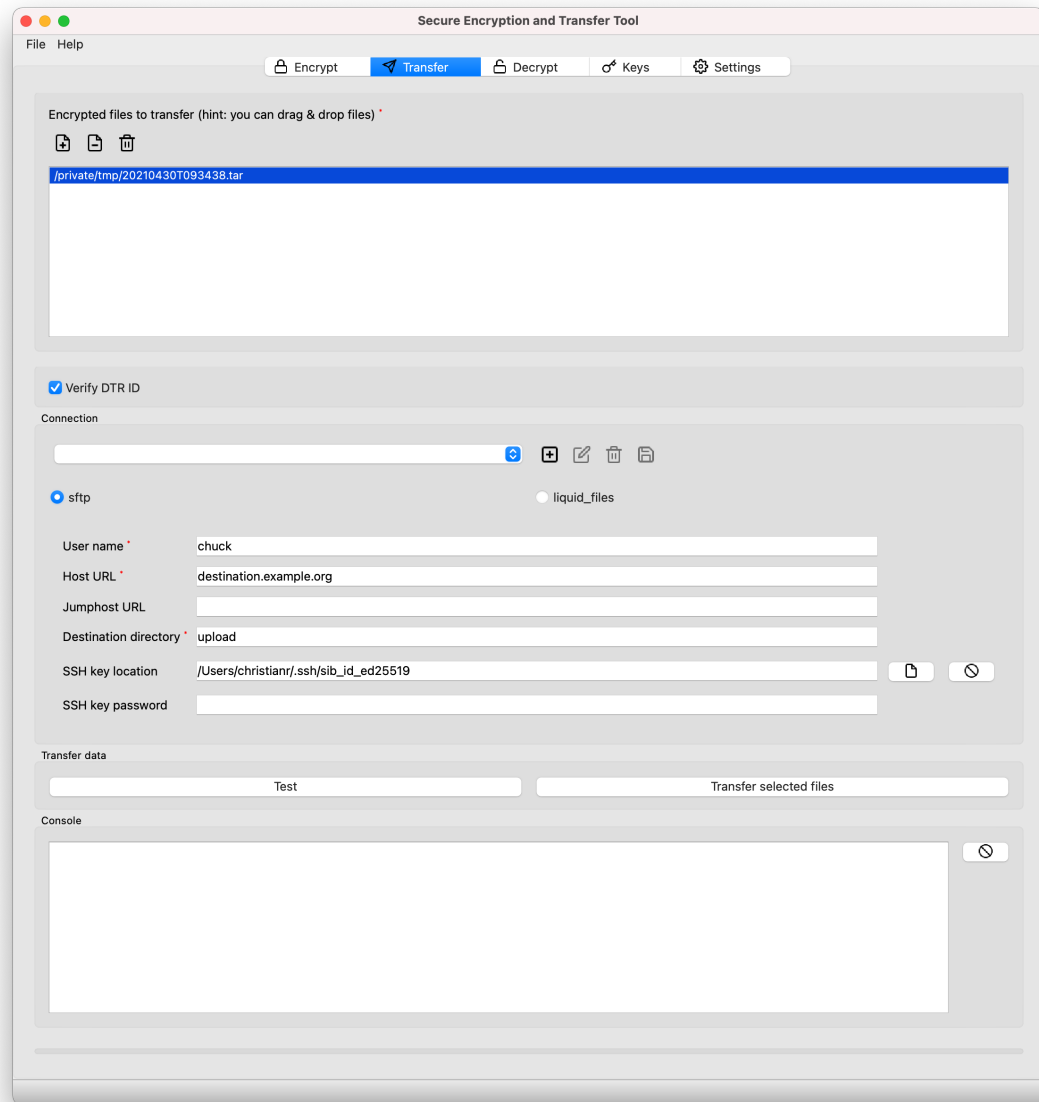
The objective of this verification is to avoid that users mistakenly include sensitive information in data package file names.

- **Verify key approval:** Verify that the PGP keys (sender and recipients) have been approved by the central authority.

### 6.2.1 Transferring files with sett-gui

To transfer encrypted files:

1. Go to the **Transfer** tab of the *sett* application.



2. **Select encrypted files to transfer:** click **Add files** and select a .zip file that was generated using the *sett* application.
  - Multiple files can be transferred at the same time by adding more than one file.
  - Only .zip files produced by the *sett* application can be transferred.
3. Select the data transfer **Protocol** (**sftp**, **s3** or **liquid\_files**). The choice depends on the server to which the data should be sent, but in most cases **sftp** is used.

4. Set the connection parameters for the transfer:

- **User name:** the user name with which to connect to the SFTP/liquid files server.
- **Host URL:** URL address of the server where the files should be sent.
- **Destination directory:** absolute path of directory where files should be saved on the server.
- **SSH key location:** name and full path of the private SSH key used for authentication to the SFTP server. This is only required if the SSH key is in a non-standard location. Only RSA keys are accepted.
  - Do not confuse SSH keys - which are used to authenticate yourself when connecting to an SFTP server during file transfer - with PGP keys - which are used to encrypt and sign data.
- **SSH key password:** password associated with the private SSH key given under **SSH key location**. If your SSH key password contains characters that are *not ASCII characters*, and that this results in an error, please see the *SSH private key with non-ASCII characters* section of this guide.

---

### BioMedIT

For BioMedIT users, the SFTP connection parameters **User name**, **Host URL**, and **Destination directory** will be provided by your local BioMedIT node.

---

5. You are now ready to transfer the data. Click **Transfer selected files** and follow the progress of the transfer using the progress bar and the **Console** box.

## 6.2.2 Transferring data on the command line

*sett* command to transfer data:

```
# General syntax:
sett transfer --protocol sftp --protocol-args '{"host": "HOST", "username": "USERNAME",
↪ "destination_dir": "DIR", "pkey": "PRIVATE_RSA_SSH_KEY"}' FILES_TO_TRANSFER
# 'session_token' is STS credentials specific
sett transfer --protocol s3 --protocol-args '{"host": "localhost:9000", "secure": 1,
↪ false, "bucket": "BUCKET", "access_key": "ACCESS_KEY", "secret_key": "SECRET_KEY",
↪ "session_token": "SESSION_TOKEN"}' <files to transfer>
sett transfer --protocol liquid_files --protocol-args '{"host": "HOST", "subject":
↪ "SUBJECT", "message": "MESSAGE", "api_key": "APIKEY", "chunk_size": 100}' FILES_TO_
↪ TRANSFER

# Example:
sett transfer --protocol sftp --protocol-args '{"host": "10.0.73.1", "username": "alice",
↪ "destination_dir": "/data", "pkey": "~/.ssh/id_rsa"}' encrypted_data.zip
```

Note that if you are using the **Windows Command Prompt**, the above syntax must be modified to use double quotes (") instead of single quotes ('), and doubled-double quotes (") instead of regular double quotes (") around the `--protocol-args`. Here is an example:

```
# Example for Windows command prompt:
sett transfer --protocol sftp --protocol-args "{""host"":""10.0.73.1"", ""username"":""
↪ alice"", ""destination_dir"":""/data"", ""pkey"":""~/.ssh/id_rsa""}" encrypted_
↪ data.zip
```

For SFTP transfers, an **SSH key** is required for authentication on the host server. The private SSH key can be provided via 2 mechanisms:

- Specifying the location of the key via the `pkey` argument of `--protocol-args`. See below for more details.
- Use an SSH agent to provide the key. The SSH agent is automatically detected by *sett* and no specific input from the user is needed. In this case, the `pkey` argument should be skipped.



The `--protocol-args` value takes a different set of fields depending on the protocol being used:

- **sftp** protocol arguments:
  - **host**  
Address of remote SFTP server to which files should be transferred, e.g. `"host": "10.0.73.1"`.  
Connecting to a specific port on the SFTP server can be done using the syntax: `"host": "10.0.73.1:3111"` (here 3111 is the port to use). If no port is specified, port 22 is used by default.
  - **username**  
User name with which to connect to the SFTP service.
  - **destination\_dir**  
Absolute path of the directory on the SFTP server where files should be transferred.
  - **pkey**  
Path + name of the private SSH key used to authenticate with the SFTP server. This argument is only needed if the authentication method used by the SFTP server is SSH key (not needed with OIDC).  
  
This argument can also be skipped if the SSH key is provided via an SSH agent - i.e. if `pkey` is missing, *sett* will try to find a suitable SSH key from a running agent.
  - **pkey\_password**  
If a private SSH key is passed via the `pkey` argument, its password should be provided via this argument. If the key is not password-protected (unsecure - not recommended), `"pkey_password": ""` must be passed. If this argument is missing, the user will be manually prompted to enter the password. If your SSH key password contains non-ASCII characters, and that this results in an error, please see the [SSH private key with non-ASCII characters](#) section of this guide.
- **liquid\_files** protocol arguments:
  - **host**  
Address of remote liquid files server to which files should be transferred.
  - **api\_key**  
The API key from liquid files. You can get from your account: “Account Settings” -> “API”.
  - **subject**  
Optional subject to send together with the package.
  - **message**  
Optional message to send together with the package.

Adding the `--dry-run` option will run the `transfer` command in test mode, i.e. checks are made but no data is transferred.

To display help for the transfer command: `sett transfer --help`.

## 6.3 Decrypting files

The *sett* application allows the decryption and decompression of files in a single step. However, **only files encrypted with the sett application, or files that follow the [sett packaging specifications](#) can be decrypted with sett**.

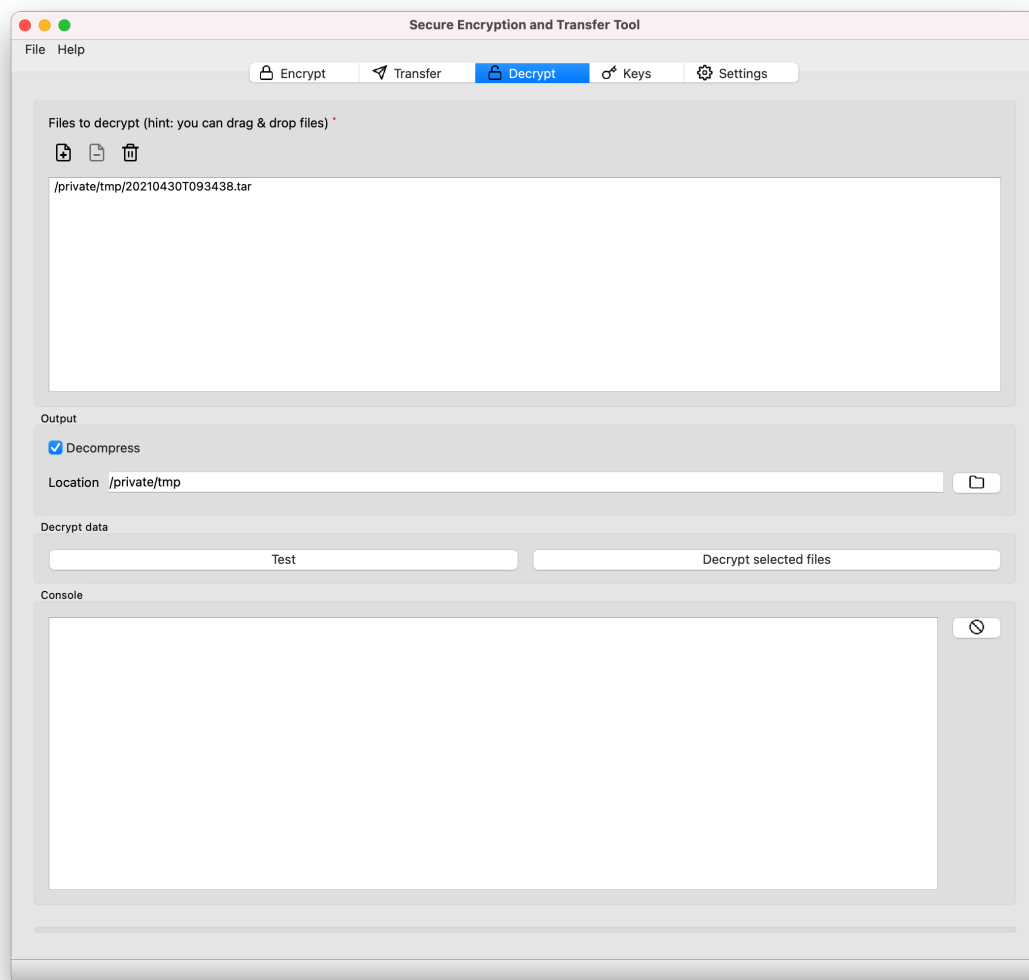
The following OpenPGP certificates **must be present in the local sett certificate store** in order to decrypt data:

- The public certificate of the data sender (i.e. the person who encrypted the data). This is needed to verify the authenticity of the data package.
- The secret certificate of the data recipient (the person for whom the data has been encrypted).

### 6.3.1 Decrypting data with sett-gui

To decrypt and decompress files:

1. Go to the **Decrypt** tab of the *sett* application (see figure below).



2. Select one or more files to decrypt with **Add files**.
3. Selecting **Decompress** will both decrypt and decompress files. Deselecting this option will only decrypt the package, outputting a compressed file named `data.tar.gz` that contains the decrypted data.
4. **Location**: select a location where to decrypt/decompress files.
5. Click **Decrypt selected files** to start the decryption/decompression process. A pop-up dialog box will appear asking for the password associated with the PGP key used to encrypt the files.

### 6.3.2 Decrypting data on the command line

*sett* command to decrypt data:

```
# General syntax:
sett decrypt --output-dir=OUTPUT_DIRECTORY ENCRYPTED_FILES.zip

# Example:
sett decrypt --output-dir=/home/alice/data/unpack_dir /home/alice/data/test_data.zip
```

To display help for the decrypt command: `sett decrypt --help`.

To decrypt data without decompressing it, add the `--decrypt-only` option.

If the `--output-dir` option is omitted, the data is decrypted in the current working directory.

If you want to automate the decryption process you can use the `--passphrase-cmd` option with an external command that returns your PGP key password to the standard output.

**IMPORTANT: make sure the external command and the password store are secure.**

## 6.4 PGP key auto-download and auto-refresh in *sett*

To ensure that all PGP keys used for data encryption and signing are trustworthy and up to date, *sett* is configured by default to work in conjunction with the **OpenPGP keyserver** [keys.openpgp.org](https://keys.openpgp.org) and the **BioMedIT portal**. If you are using *sett* outside of BioMedIT, you will need to disable the usage of the BioMedIT portal via the *sett* [configuration options](#).

The **keyserver** is an online server where users upload their public PGP keys, so that people who need these keys - to encrypt data or verify a signature - can easily obtain them.

PGP keys used by *sett* (data recipients and sender keys) are automatically refreshed from the keyserver each time they are about to be used. This ensures that an up-to-date version of the keys is used. For instance, if a key is revoked, then the user's local copy of the key will be automatically updated with this information.

If needed, the auto-download/auto-refresh of PGP keys can be disabled via the **Allow PGP key auto-download** checkbox in the **Settings Tab**.

The **BioMedIT portal** is an online service where users can (among other things) register their PGP key and get it approved for usage within BioMedIT. Just before a key is used, *sett* connects to the portal to verify that it has been approved.

For non-BioMedIT users, key approval should be disabled via the **Verify key approval** checkbox in the **Settings Tab**.

---

#### BioMedIT

**Allow PGP key auto-download** and **Verify key approval** must be enabled.

---

## 6.5 Setting up predefined connection profiles for frequent use

To avoid retyping connection settings for every transfer, connection profiles can be saved in *sett*.

You can configure and save connection profiles using the *Transfer tab* of the *sett-gui* interface.

Alternatively, you can modify the *sett configuration file*. An example config file would look like this:

```
{
  ...

  "connections": {
    "custom_connection": {
      "protocol": "sftp",
      "parameters": {
        "destination_dir": "upload"
        "host": "server.name.com",
        "pkey": "",
        "username": "chuck_norris",
      }
    }
  }
}
```

Predefined connections can then be selected from the **Connection** drop-down menu in the **Transfer** tab of the *sett-gui* interface.

On the command line, predefined connections can be passed using the `--connection` option, as illustrated here. In that case, the `--protocol` and `--protocol-args` are no longer needed, unless you wish to override part of the predefined connection's settings:

```
# General syntax:
sett transfer --connection PREDEFINED_CONNECTION_NAME FILES_TO_TRANSFER

# Example:
sett transfer --connection custom_connection encrypted_data.zip
```

## 6.6 Working with large files

*sett* does not support multi-threading during file compression/decompression, and therefore the default **encryption** and **decryption** workflows can take a long time when working with large files (> 100 GB).

To alleviate this problem, *sett* provides the option to skip the compression/decompression steps in both the encryption and decryption workflows. Users can then compress/decompress their data with an external compression tool that supports multi-threading (e.g. *pigz* on Linux or *7zip* on Windows), before encrypting (or after decrypting) it with *sett*'s compression/decompression option disabled.

Disabling compression/decompression in *sett-gui*:

- To disable compression in the encryption workflow, uncheck the **Compress input data** checkbox in the **Encrypt** tab.
- To disable decompression in the decryption workflow, select **Decrypt only** in the **Select data decryption options** section of the **Decrypt** tab.

Disabling compression/decompression in *sett* command line:

- To disable compression in the encryption workflow, add the `--compression-level=0` option to the *sett* encrypt command.

- To disable decompression in the decryption workflow, add the `--decrypt-only` option to the `sett decrypt` command.
- Examples:

```
# Data encryption with compression disabled.
sett encrypt --sender alice.smith@example.com --recipient bob@example.com --dtr-
→id 42 --compression-level=0 --output test_output ./test_file.txt ./test_
→directory

# Data decryption with decompression disabled.
sett decrypt --decrypt-only --output_dir=/home/alice/sett_demo/unpack_dir /home/
→alice/sett_demo/test_data.zip
```

### 6.6.1 Split and transfer large files

Transferring a single large files can sometimes prove problematic, e.g. if the connection is unstable. When encountering such problems, one workaround is to split the data to encrypt and transfer into smaller chunks.

#### Windows - using 7-zip

On Windows, the open source utility *7zip* can be used to compress and split data.

- With *7zip* installed, right-click on the folder you want to compress and split: select 7-Zip > Add to archive....
- In the *7zip* dialog box, set the following values:
  - Click on ... to change the name and location of the output archive file.
  - Set Archive Format to zip.
  - Set Split to volumes, bytes to an appropriate size in Megabytes (M) or Gigabytes (G), e.g. 100G for 100 gigabytes chunk size.
- The output filenames will be numbered like this: `archive.zip.001`, `archive.zip.002` etc.
- Encrypt these archive files using *sett* without additional compression, see instructions above.

Once data is transferred and decrypted at the destination, it can be inflated again. This can be done with the `unzip` command line utility:

- `unzip archive.zip.0xx`, where `0xx` is the last file, i.e. the file with the highest number. It contains the file structure.

*Note:* *7zip* can also be used in command line.

- `"C:\Program Files\7-Zip\7z.exe" a -v100g "archive.zip" "folder/to/be/archived"`
- The `-v100g` option creates 100 Gigabytes chunks (volumes).
- Several files/directories to compress can be specified after `"archive.zip"`, and asterisks `*` can be used to match multiple files, e.g. `"data/*.csv"`.

## Mac OS and Linux - command line

On **Mac OS** and **Linux**, the `zip` command line utility can be used to split and compress data.

- In the terminal, `cd` to the directory containing the files/directories to compress.
- Run the command: `zip -s 100g archive.zip <file(s) or directory(ies)>`. In this example, data will be split into 100 GB chunks, but the size of chunks can be set as appropriate by replacing `100g` with the desired value.
- This will create numbered files named `archive.zip`, `archive.z01`, `archive.z02`, etc. The `archive.zip` is the last file created and contains the file structure.
- Encrypt these archive files using *sett* without additional compression, see instructions above.

Once data is transferred and decrypted at the destination, it can be inflated using the command `unzip archive.zip`.

## Mac OS with GUI - using Keka

On **Mac OS**, people preferring to use a GUI than command line can use [Keka](#), a compression app with Finder integration. The app is 5 CHF via Mac the App Store or free via their website (donations welcome).

- Open Keka, and specify the split size in Megabytes or Gigabytes, e.g. `100 GB`.
- In the *Settings* menu, activate the Finder integration.
- Right-click on a directory to compress it.

## 6.7 Automating tasks with sett

For users that need to package and transfer data on a regular basis, a number of options exist to simplify the automation of *sett* tasks. Please refer to the [sett automation section](#) for details.

## 6.8 Installing or running sett behind a proxy

If you are running *sett* behind a proxy, the shell environment variable `ALL_PROXY` or `HTTPS_PROXY` must be set. This is the recommended and global way to specify a proxy. Note that, while certain programs support a *proxy* option (e.g. `pip` with `--proxy`), there is currently **no such option in sett**.

**Example:**

```
ALL_PROXY=https://host.domain:port sett-gui
```

---

**Note:** If your proxy is a socks proxy, you need to install *sett* with socks proxy support:

```
pip install [ --user ] sett[socks]
```

In this case also replace the schema `https://` with `socks5://`.

---

### 6.8.1 Checking whether you are using a proxy

On Windows, you can check if you are using a proxy server to access the internet by going to **Start > Settings > Network & Internet > Proxy** (Windows 10). If the slider under **Use a proxy server** is “off”, no proxy is being used.

If you are told that you need to set a proxy, input the **Address and Port** details and click **Save**. If in doubt please consult with your IT department.

On Mac OS the proxy information is located under the **System Preferences > Network > Advanced > Proxies** tab of the network interface, usually Ethernet or Wi-Fi.

## 6.9 Known issues and limitations

### 6.9.1 SSH private key with non-ASCII characters password

Even though it is possible to create an SSH key pair using a password containing non-ASCII characters, it seems like those characters are encoded differently between different operating systems.

As an SSH key might be moved to a machine with another operating system, or encoding might change with a new version, it is impossible to guess the correct encoding in any cases. **For this reason, we recommend not to use non-ASCII characters to protect SSH private keys.**

If this is still desired, there is a configurable option `ssh_password_encoding` available in the `sett config` file which defaults to `utf_8` (this is correct encoding for keys generated with `ssh-keygen` on linux / mac). For keys generated with `ssh-keygen` on Windows 10, `cp437` should work to correctly encode non-ASCII chars. Example of config file with SSH password encoding set to `cp437`:

```
{  
  "ssh_password_encoding": "cp437"  
}
```





## AUTOMATING SETT

### 7.1 Batch packaging and transferring of files

While *sett* does not offer “batch” processing natively, you can use *pkg-catapult*, a companion tool that automates data packaging and transfer.

Specifically, **pkg-catapult** has a “batch” mode that allows to specify different sets of files to package and transfer via a simple text file input. The term “batch” processing here refers to the packaging and transfer of multiple file sets as independent packages - *sett* can already natively package multiple files in a single package.

Please refer to the **pkg-catapult** documentation for more details.

### 7.2 Dealing with passwords when automating sett tasks

A difficulty in automating *sett* tasks is the handling of passwords that are required during data encryption (password for the private PGP key used to sign packages) and transfer (password for the private SSH key used to connect to the SFTP server where data is being transferred).

The following section offers some guidance on how to deal with passwords when automating *sett* tasks.

**Warning: Security warning:** automating a workflow that involves secrets (i.e. the need to enter a password) almost always means **weakening the security chain**. Please consider the following points carefully before starting to automate tasks with *sett*:

- Against whom should the data be protected?
- Is the protection sufficient for that purpose on both ends of the communication channel?

#### 7.2.1 Example 1: Fully automated - unencrypted password file

Given:

- The following connections section in the *sett configuration*:

```
{
  "connections": {
    "area51": {
      "parameters": {
        "host": "area51.org",
        "username": "chucknorris",
        "destination_dir": "upload",
        "pkey": "/home/chucknorris/.ssh/pkey",
        "pkey_password": ""
      }
    },
  },
}
```

(continues on next page)

(continued from previous page)

```
    "protocol": "sftp"
  }
}
```

- An unprotected private ssh key `/home/chucknorris/.ssh/pkey`.
- A gpg key protected with the password contained in a file `.pw`.

Then, the following will automate packaging and upload (and could potentially be put into a cron job):

```
#!/bin/bash

sett encrypt -r bob@example.com --dtr-id 42 --purpose PRODUCTION -o output ./files --
↪passphrase-cmd "cat .pw"
sett transfer --connection area51 output/*
```

### 7.2.2 Example 2: Semi automated

- Set `"pkey_password": null` in the above example.
- Use a password protected SSH private key instead (still `/home/chucknorris/.ssh/pkey`)
- As above, use a PGP key protected with the password contained in a file `.pw`.
- Before activating the cron job / the shell script, load the ssh key into the ssh agent (have to repeat this on every login to the machine):

```
ssh-add /home/chucknorris/.ssh/pkey
```

Then activate a cron job / invoke the above commands.

## SETT CONFIGURATION FILE

*sett* allows a number of options to be customized via its configuration file. For instance, you may change the default compression level, or define a default data sender.

Depending on your operating system, the default location for the **sett configuration file** is:

- Linux: `~/.config/sett/config.json`
- Windows: `C:\Users\%username%\AppData\Roaming\sett\config.json`
- Mac OS: `~/.config/sett/config.json`

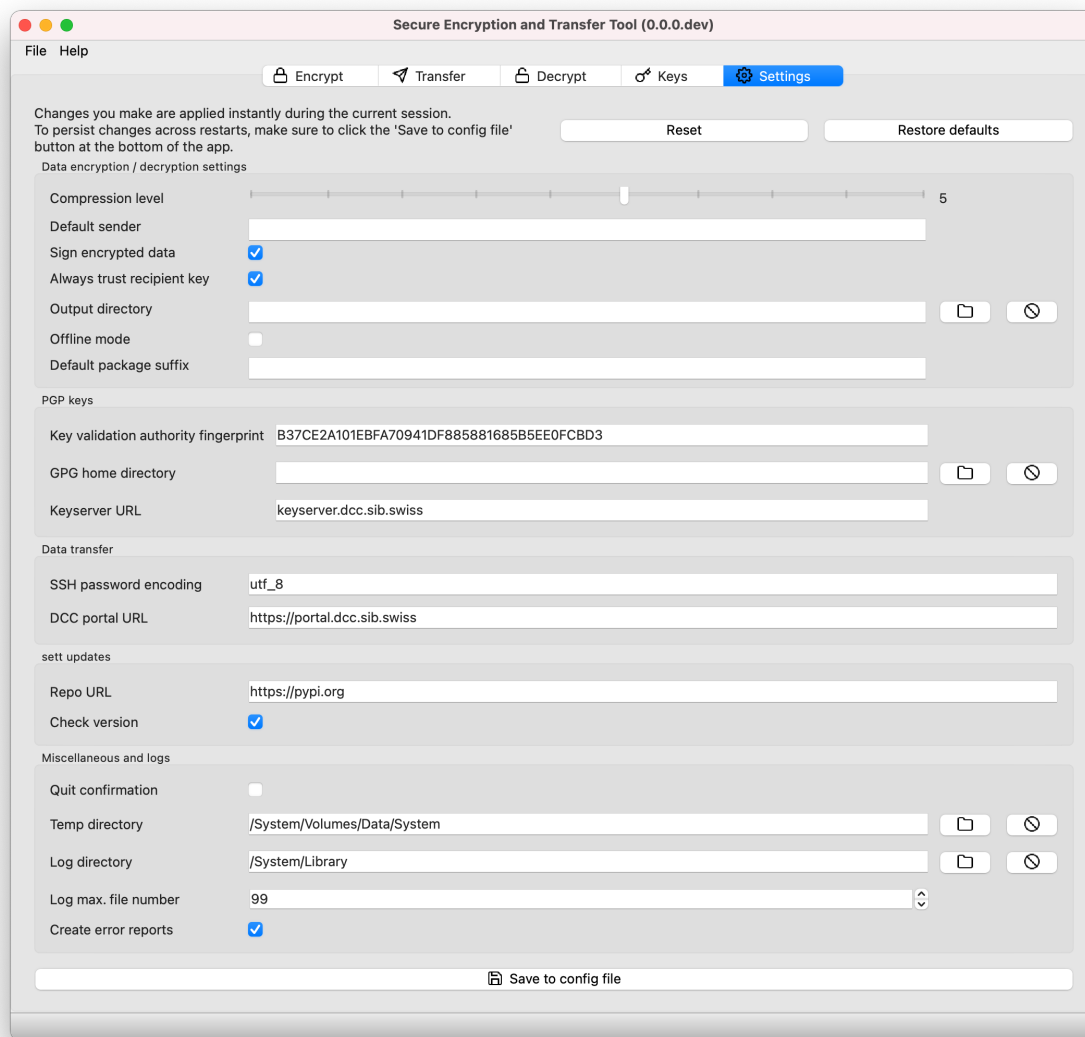
Each configuration option has a predefined default value in *sett*, which is used in case the option is missing from the config file, or if there is no config file. This means that not all configuration options need to be specified in the config file, but only those for which the default value should be overridden.

The *sett* configuration settings can be modified either by directly editing the config file, or by using the `sett-gui` application.

Configuration settings that refer to paths (`gpg_home_dir`, `log_dir`) support the usage of `~` as a shortcut for a user's home directory in the path. When *sett* writes or overwrites a config file, it will automatically use the `~` shortcut so that the file becomes more easily portable to a different machine.

### 8.1 Configuration changes using *sett-gui*

The easiest way to change the *sett* configuration settings is by using the **Settings** tab in `sett-gui`:



Any change made in the **Settings** tab becomes effective immediately. However, to make changes persistent (i.e. so that changes are kept even if the application is closed), the **Save to config file** button at the bottom of the tab must be clicked.

Clicking on **Save to config file** will write the current settings to the config file, making them persistent across restarts of the application.

Resetting changes made in the **Settings** tab back to the last persisted settings (i.e. the values in the config file) can be done by clicking on the **Reset** button.

Settings can also be reset to their factory defaults by using the **Restore defaults** button, e.g. if your config file has become corrupted.

For detailed settings explanations, please refer to the [configuration options section](#) below.

## 8.2 Configuration changes using command line

If no *sett* configuration file is present on your system, it can be created with:

```
sett config --create
```

To display your current configuration settings, the following command can be used. Note that the shown values are the combination of the default values, overridden with the values read from the config file (if a config file exists).

```
sett config --show
```

For detailed explanations on the different settings, please refer to the *configuration options* section below.

## 8.3 Alternative config file location

The `SETT_CONFIG_FILE` environmental variable allows specifying an alternate location for the *sett* configuration file. In the example below, the config file location is changed to `/home/user/custom_sett_config.json`:

```
# Setting a custom config file location for a single command run in a UNIX bash shell.
SETT_CONFIG_FILE=/home/user/custom_sett_config.json sett ...
SETT_CONFIG_FILE=/home/user/custom_sett_config.json sett-gui

# Setting a custom config file location for an entire UNIX bash shell session.
export SETT_CONFIG_FILE=/home/user/custom_sett_config.json
sett ...
sett-gui

# Setting a custom config file location in a Windows PowerShell.
# Note: under Windows there should be no quotes around the variable value.
#       E.g. setting SETT_CONFIG_FILE="C:\Users\Alice\sett_config.json" will not work.
set SETT_CONFIG_FILE=C:\Users\Alice\Documents\custom_sett_config.json
sett ...
sett-gui
```

## 8.4 Configuration options

The following options can be set in the config file:

### **always\_trust\_recipient\_key**

If false, the encryption key must be signed by the local user (true or false).

### **check\_version**

Check whether you are running the latest version of *sett* on start-up (true or false).

### **compression\_level**

Default compression level to be used by *sett* when creating new encrypted data packages (reminder: the data is compressed before it is encrypted). Values must be an integer number in the range 0 (no compression) to 9 (maximum compression). Higher compression levels require more computing time.

### **connections**

List of predefined connections for data transfer. Example:

```
"connections": {
  "sftp@localhost": {
    "parameters": {
      "destination_dir": "upload",
```

(continues on next page)

(continued from previous page)

```
    "host": "localhost",
    "pkey": "~/.ssh/sftp_ssh_testkey",
    "username": "sftp"
  },
  "protocol": "sftp"
}
```

**dcc\_portal\_url**

URL of a **BioMedIT portal** instance. The **portal** is used for key approval and **DTR** (Data Transfer Request) validation.

**default\_sender**

Default sender fingerprint for encryption.

**gpg\_home\_dir**

Path of the directory where **GnuPG** stores its keyrings and other configuration settings.

**gui\_quit\_confirmation**

Ask for confirmation to quit *sett-gui* (true or false). This option only applies to GUI and does not affect CLI.

**gpg\_key\_autodownload**

Allow the automatic download and refresh of PGP keys from the keyserver (true or false). If true (default value), PGP keys used during encryption/decryption are attempted to be refreshed from the keyserver before they are used (provided a value for `keyserver_url` is given).

**keyserver\_url**

URL of the keyserver: used for publishing/fetching public PGP keys.

**legacy\_mode**

When enabled, *sett* reverts to using **GnuPG** as cryptographic backend instead of the **Sequoia** library. In legacy mode, *sett* will also look for PGP keys in the GnuPG keyring, instead of its own certificate store.

**log\_dir**

Path to log files directory. By default, the log files are saved at the following locations (~ indicates the user's home directory):

- Linux: `~/.local/var/log/sett`
- Mac OS: `~/.local/var/log/sett`
- Windows: `~\AppData\Roaming\sett`

**log\_max\_file\_number**

Maximum number of log files to keep as backup.

**max\_cpu**

Maximum number of CPU cores for parallel computation. Use 0 (default) if you want to use all available CPU cores. Note, only some parts of the encryption and decryption workflows can parallelize computation.

**output\_dir**

Default output directory, relevant for encryption/decryption. This option only applies to *sett-gui*. In command line, the output location can be specified via the optional argument `--output` (resp. `-o`) for encryption and `--output-dir` (resp. `-o`) for decryption.

**package\_name\_suffix**

Default suffix for encrypted package name. This option applies to both, *sett-gui* and the command line. In command line it is possible to override the default suffix with the optional argument `--output-suffix`.

**repo\_url**

Python package repository, used when looking for *sett* updates.

**sftp\_buffer\_size**

Size of buffer in bytes when transferring data with the SFTP protocol (default: 1048576 bytes = 1 Megabyte).

This is the size of every packet sent over the network. Larger buffer sizes theoretically result in faster transfer speeds on stable networks but will decrease transfer speed on poor networks (networks with higher probability of packet losses). The default value is on the optimistic side, you might need to decrease it if transferring data on poor quality networks.

**ssh\_password\_encoding**

Character encoding used for the SSH key password. By default the encoding is assumed to be `utf_8`. See the *SSH private key with non-ASCII characters* section of this guide for further details.

**verify\_dtr**

Verify that the given Data Transfer Request (DTR) ID is valid and the associated metadata is correct, i.e.

- **DTR ID** is valid and the transfer is authorized.
- **Sender** and **Recipients** public PGP keys are approved by the BioMedIT key validation authority.
- **Recipients** are approved *Data Managers* of the BioMedIT project for which data is being encrypted.

When **Verify DTR** is enabled, the project code of the project to which the DTR belongs will be added as a prefix to the output file name.

**verify\_key\_approval**

Verify that a PGP key has been approved by the central authority.

**verify\_package\_name**

If `true` (the default), the name of data packages files (i.e. the encrypted files) are verified to match the pattern `<project_code>_<date_format>_<package_name_suffix>.zip` before they are transferred. This is to verify that no sensitive information has been mistakenly included in the file name. This check can be permanently turned-off by changing the value of the setting to `false`, or by un-checking the corresponding checkbox in the Settings tab of *sett-gui*.





## SETT PACKAGING FILE FORMAT SPECIFICATIONS

*sett* compresses, encrypts and packages files in a single `.zip` file whose specifications are described below. Only files adhering to these specifications can be transferred or decrypted by *sett*, and failure to comply with the specification will generate an error.

### 9.1 File structure

*sett* `.zip` files have the following structure:

```
YYYYMMDDThhmmss.zip
├─ metadata.json
├─ metadata.json.sig
├─ data.tar.gz.gpg
│   └─ data.tar.gz
│       ├── content/
│       │   ├── [file1]
│       │   ├── [file2]
│       │   └─ ...
│       └─ checksum.sha256
```

#### **metadata.json**

Metadata file containing the following information:

- **transfer\_id**: unsigned integer indicating the transfer ID, or `null`.
- **sender**: fingerprint of the sender's public PGP key.
- **recipients**: list of fingerprints of the recipients' public PGP keys.
- **timestamp**: datetime of metadata generation. Uses the datetime format `YYYY-MM-DDThh:mm:ss±ZZZZ` (where `ZZZZ` is the UTC offset).
- **checksum**: sha256 checksum of the encrypted `data.tar.gz.gpg` file. This allows verifying the integrity of the transferred file without decrypting it, and is used to make sure nothing was corrupted during the transfer process.
- **checksum\_algorithm**: algorithm used to compute the checksum. Currently `sha256`.
- **compression\_algorithm**: algorithm used to compress the inner tarball. The following values are supported:
  - `zstandard`, for data compressed with *zstandard*. This is the data encryption algorithm used by default in *sett*.
  - `gzip`, for data compressed with *gzip*.
  - `stored`, for data that are not compressed, but simply packaged with `tar`.
- **purpose**: one of `PRODUCTION`, `TEST` or `null`.
- **version**: the version of the metadata specifications.

**metadata.json.sig**

Detached PGP signature for the metadata.json file.

**data.tar.gz.gpg**

A tarball encrypted using the receiver's public PGP key and signed with the sender's private key. It is compressed with `compression_algorithm`.

**[file1], [file2]**

One or several data files can be transferred. Data to be sent can be in any format, e.g. `.txt`, `.csv`, `.dat`.

**checksum.sha256**

sha256 checksum file of all files present in `data.tar.gz`. This is used to make sure nothing was corrupted during the encryption/transfer/decryption process.

**Important:** the `.zip` file itself must use the **stored compression** method, i.e. it should not do any compression of its content. This is because the content of the zip archive (the `data.tar.gz.gpg` file) is already compressed, so compressing it a second time is unnecessary.

## 9.2 File example

Examples of the content and structure of the metadata and checksum files.

**metadata.json**

```
{
  "transfer_id": 42,
  "sender": "AAABFBC698539AB6CE60BDBE8220117C2F906548",
  "recipients": ["D99AD936FC83C9BABDE7C33E1CF8C1A2076818C3"],
  "timestamp": "2020-01-29T15:31:42+0100",
  "checksum": "a8eb0ee5a6a53326b1c6f9bf94136da5d98a1dc6dceee21d62f694d71c4cf184
→",
  "checksum_algorithm": "SHA256",
  "compression_algorithm": "gzip",
  "purpose": "PRODUCTION",
  "version": "0.7"
}
```

**checksum.sha256**

```
41421f0c4a5353a5a0cdd37de3fd80a840a190ca997ad8044a67c4c1683f7b63 file1.csv
35ba157ed1c3269d731a438c466790a4f481bb49805e2d1f380df0c636792ff6 folder1/file.txt
fd9ebdbcc1a5fc35ded6e78a6b16ef658502c9d0b05dd4a2185d0f94ccf165cf folder1/folder2/
→file.txt
```

## SETT BENCHMARKS

### 10.1 Setting-up your system to run the sett benchmarks

1. If not already done, install **sett** following *these instructions*.
2. Download the *benchmark script* file.
3. If needed, install the required dependencies - see *Requirements*.

**Warning:** If you are installing the benchmark script as a system administrator, please be aware that it can potentially be **exploited by users to execute any shell command on a system**. Therefore the benchmark script should only be granted permission to run with regular user permissions, i.e. never give SUID permissions to the script.

#### 10.1.1 Requirements

- *Python*  $\geq 3.8$
- *Podman* if the `sftp_server` argument value is set to `podman`.
- *Docker* if the `sftp_server` argument value is set to `docker`.
- *pigz* if using external compression as part of the benchmarks (see `compression_cmd` argument in the config file). *pigz* will be needed to run the most of the presets.
- **WARNING:** the benchmarks currently **only work on Linux and MacOS X systems**.

### 10.2 Running the sett benchmarks

The *sett* benchmarks are run by executing the script `run_benchmarks.py` as illustrated below.

The benchmarks can be run with a number of different factor levels (e.g. data size, data type, *sett* commands to run, etc.) and other options, all of which are described in the *Benchmark parameters* section below.

While all benchmark parameters have a default value, some might need to be changed to match your local system configuration. You should pay particular attention to the values of the following parameters:

- `sftp_server`: defaults to `podman`, but you might need to change it (e.g. to `docker` if you are using docker to run containers).
- `root_dir`: defaults to the current working directory.

To modify a parameter from its default, its value must be set via the following config file:

- Linux: `~/.config/sett/benchmark_config.json`
- Windows: `C:\Users\%username%\AppData\Roaming\sett\benchmark_config.json`

- Mac OS: `~/ .config/sett/benchmark_config.json`

To run the benchmarks, execute the `run_benchmarks.py` file as shown in these examples:

```
./run_benchmarks.py           # run the benchmarks.
./run_benchmarks.py --dry-run  # run the script in "dry-run" mode.
./run_benchmarks.py -d --preset=full  # dry-run mode with "full" preset.
./run_benchmarks.py --preset=quick    # run the benchmarks with "quick" preset.
```

The benchmark outputs are saved in a tab-delimited file named `benchmark_results.tsv` (or `benchmark_results_<preset>.tsv` if running a preset), which is saved in the directory specified through the `root_dir` parameter.

Outside of the benchmark parameters passed via the config file, the benchmark script itself takes 2 optional arguments:

- `--dry-run / -d`: dry-run mode. Displays the parameter combinations that to be run without running the actual benchmarks.
- `--preset / -p`: run the benchmarks with a “preset” that overrides both the default and the user config file values. Three presets are available: “quick”, “short” and “full” - “q”, “s” and “f” can be used as abbreviations.

Here is a summary of each preset - to see the detailed values associated with each preset, run it in `--dry-run` mode:

- **quick**: preset to do a quick check of whether the benchmark script is working on your machine. It runs only small input data sizes and takes a few minutes to complete.
- **short**: preset that runs only a limited number of parameter combination, but with input data sizes up to 100 GB. Please note that **running this preset will require ~200 GB of free disk space on your machine**. The total runtime for this preset can be **up to 30 hours**.
- **full**: preset that runs the full combination of possible parameter values for input data up to 1 GB. Running this preset will require **~ 10 GB of free disk space** on your machine. The **total runtime for this preset can be up to 4 days**.

## 10.3 Benchmark results plotting

Automated plotting of benchmark results is possible **for a specific set of presets** using the `plot_benchmarks.py` script.

Please note that the `plot_benchmarks.py` script only works to plot the results from the `short` and `full` presets. By default the input files are expected to be named `benchmark_results_short.tsv` and `benchmark_results_full.tsv`, but custom names can be passed via the `-s` and `-f` arguments. Run `plot_benchmarks.py --help` for more details.

If an input file is missing, the plots to be drawn for that file will simply be skipped, but the script will still run.

The set of commands to plot benchmark results is thus the following:

```
./run_benchmarks.py --preset=short
./run_benchmarks.py --preset=full
./plot_benchmarks.py
```

## 10.4 Benchmark parameters

### 10.4.1 Benchmark factor levels

The following parameters accept one or more values (in a list format) that are used as factor levels in the benchmarking. The benchmarks will be carried-out across all combinations of the different parameter values.

**data\_size:**

Total size of data in megabytes [MB] to be used as input for the *sett* “encrypt” workflow. The minimum value is 1 MB and the maximum value depends on how much space is available on the machine running the benchmarks. Please note that the amount of available disk space must be at least twice the maximum value.

Example: "data\_size": [100, 200, 500, 1000, 5000, 10000, 20000, 100000]

**file\_size\_type:**

The file size composition of the data used for the benchmarking. Possible values are:

- "small": the data to encrypt with *sett* consists exclusively of a combination of small (1 MB) files.
- "large": the data to encrypt with *sett* consists of a combination of large files. The size of the files depends on the total data size of a given benchmark run.
- "mixed": the data to encrypt with *sett* consists of a combination of small (1 MB) and large files.

Example: "file\_size\_type": ["small", "large", "mixed"]

**file\_data\_type:**

The content of the files to encrypt with *sett* during the benchmarks. Possible values are:

- "text": use random FASTA-format like files as input.
- "binary": use random binary files as input.

Example: "file\_data\_type": ["text", "binary"]

**sett\_cmds:**

The *sett* commands to benchmark. Possible choices are "encrypt", "decrypt" and "transfer". The "encrypt" command is mandatory and will automatically be added if missing.

Example: "sett\_cmds": ["encrypt", "decrypt", "transfer"]

**compression\_levels:**

Levels of gzip/bzip2 data compression used in the *sett* “encrypt” workflow (or during external compression). Values must be in the range 0 (no compression) to 9 (highest compression).

Example: "compression\_levels": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**compression\_cmd:**

Compression method to use when packaging data. Possible values are:

- "sett": use *sett* native compression.
- "pigz": use a combination of tar and pigz to compress data. This compression method supports multi-threading.
- "7zip": not supported yet.

Example: "compression\_cmd": ["sett", "pigz"]

**compression\_cpu:**

Number of CPUs to be used for with the compression command(s). This parameter is only used when external compression is performed, since *sett* itself always uses a single CPU for data compression.

Default value: dependent on number of CPUs available on host machine.

Example: "compression\_cpu": [1, 4, 8, 16]

**replicates:**

Number of replicates to run for each benchmark combination.

Default value: 5

Example: "replicates": 3

## 10.4.2 Input/Output file locations

**root\_dir:**

Root working directory. Base working directory to be used for the benchmarking. All input and outputs will be written exclusively within this directory.

Default value: current working directory/sett\_benchmarks

Example: "root\_dir": "~/sett\_benchmarks"

**sett\_exec:**

Name of the *sett* executable to use during benchmarks. If the executable is not part of the default search PATH, the full path to the executable must be given.

Default value: the default *sett* executable present in the search PATH.

Example: "sett\_exec": ~/.local/bin/sett

## 10.4.3 SFTP parameters

**sftp\_server:**

SFTP server to use when running the *sett* “transfer” command. Possible values are:

- "podman": run a local, containerized, SFTP server using **podman**. Requires [Podman](#) to be installed on the local machine.
- "docker": run a local, containerized, SFTP server using **docker**. Requires [Docker](#) to be installed on the local machine.
- "local": files are transferred via SFTP to the local host.

Default value: "podman"

Example: "sftp\_server": "local"

**sftp\_username:**

SFTP authentication user name. User name to be used when connecting to the SFTP server. Only used when SFTP server is set to "local".

Default value: the \$USER environmental variable.

Example: "sftp\_username": "alice"

**sftp\_ssh\_key:**

Path and name of SSH private key to be used for SFTP authentication. This argument is only needed when *sftp\_server* is set to local.

Default value: ~/.ssh/id\_ed25519 or ~/.ssh/id\_rsa (whichever is present).

Example: "sftp\_ssh\_key": "~/tests/tests\_ssh\_key\_rsa"

**sftp\_ask\_for\_password:**

This parameter is only relevant when *sftp\_server* is set to "local". If *true* (the default), you will be asked to enter the password for the private SSH key used to connect to the local host. If your SSH key is does not have a password and you do not wish to be asked to enter an empty password each time, change the parameter value to *false*.

Default value: *true*

Example: "sftp\_ask\_for\_password": *false* (boolean values are unquoted and lower case)

**sftp\_destination\_dir:**

Destination directory for SFTP data transfer. The path on the SFTP server to where transferred data should get uploaded.

Default value: `"/upload"`

Example: `"sftp_destination_dir": "/upload"`

## 10.4.4 Misc parameters

**show\_stdout:**

If `"True"`, the standard output of the *sett* commands being run during the benchmarks is printed to the terminal. Setting the value to `"false"` (the default) gives a cleaner output but individual runs of *sett* commands cannot be followed as they progress.

Default value: `false`

Example: `"show_stdout": true` (boolean values are unquoted and lower case)

## 10.5 Benchmark results and sett performance

---

**Important:** All results presented here were obtained by using input data in the form of text files containing simulated [genomic data in FASTA format](#). Data compression efficiency, both in the amount of compression and the time needed for compression, can significantly vary depending on the type of input data, and therefore the *sett* performance values illustrated in this section are not representative of all data types.

---

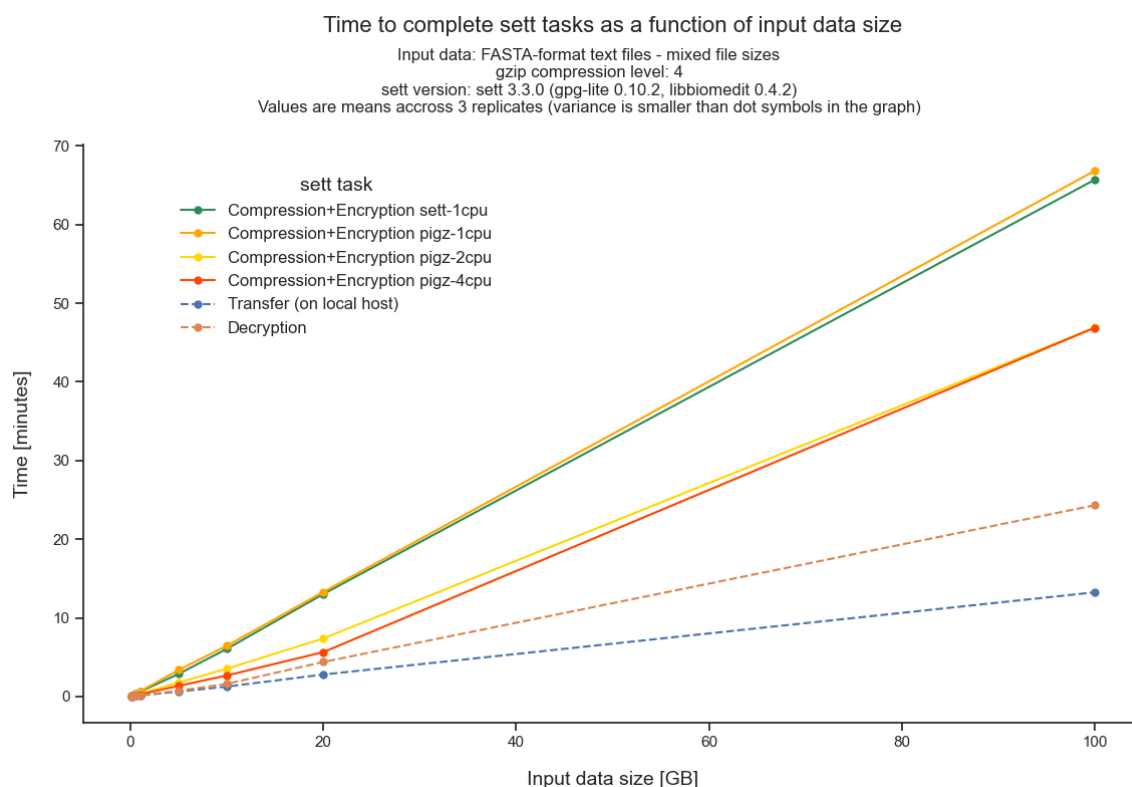
### 10.5.1 Overall sett performance

This section shows the time needed for the different *sett* tasks - data packaging (compression + encryption), transfer and decryption - as a function of input data size. If your input data is roughly similar to genomic FASTA files (or has the same compression behavior), then the graph below can be used to estimate how much time the different *sett* tasks will take.

Input data sizes up to 100 GB were tested, and, as can be seen, the time to complete the different tasks scales linearly with input data size. For data packaging (compression + encryption), *sett* is able to process data at a **speed of about 25 MB/second (1.5 GB/minute)** (CPU 2.2 GHz base and 3 GHz turbo).

Higher data compression speeds can be achieved by using an external data compression command that supports multi-threading (in our benchmark multi-threaded compression is done with `gzip + pigz`), and using *sett* only for data encryption and transfer. This can save time when processing large amounts of data.

All values shown in the figure below are averages of 3 replicated runs. The variance between runs is not shown as it is small (smaller than the dot symbol used to represent a data point). The benchmark was run on an **AWS EC2 t3a.large** instance (4 vCPUs - AMD EPYC 7571 @ 2199.742 MHz, 16GB Memory, 500GB General Purpose SSD - gp2).



*Important note regarding transfer speed interpretation:*

- Data transfer speeds via SFTP are highly dependent on the infrastructure connecting the data sender and recipients. In these benchmarks, the data recipient was an containerized (emulated) SFTP server located on the same machine that was sending the data. The transfer speeds showed here are therefore a best-case scenario.
- Please keep in mind that the values on the x-axis are input data size, and not compressed data size. Since, for our input data, the data size is roughly reduced to a third of its initial size after *sett* packaging, the actual transfer speeds are ~3 times slower than what would be concluded by directly interpreting the figure's values.

## 10.5.2 Compression level comparison

To perform data compression during *data encryption*, *sett* can use different *levels of data compression*. The data compression levels range between 0 (no compression) and 9 (maximum compression): higher compression levels produce more compressed data (i.e. smaller output file) but require higher CPU usage - and hence more time (given a fixed number of CPUs) - to do so.

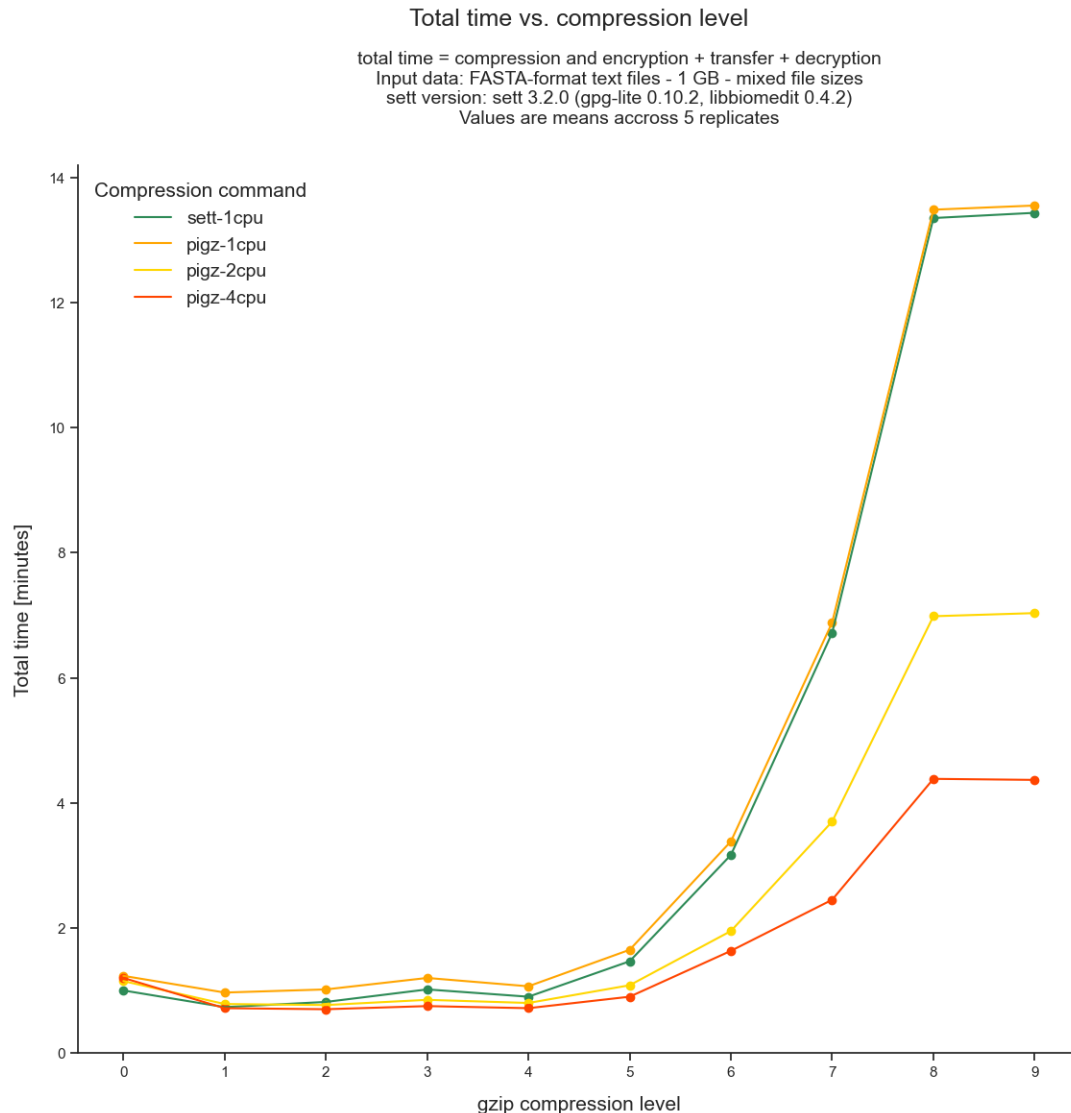
This section of the benchmark results aims to illustrate which compression level offers the best trade-off. All benchmarks in this section were run on an **AWS EC2 t3a.large** instance (4 vCPUs - AMD EPYC 7571 @ 2199.742 MHz, 16GB Memory, 500GB General Purpose SSD - gp2).



## Total data packaging, transfer and decryption time as a function of compression level

Using higher compression levels requires more time to package the data, but also results in smaller data packages, which are in turn faster to transfer. To identify the optimal compression level in this trade-off, we show here the total time needed to package, transfer and decrypt (end-to-end processing) a given size of input data as a function of compression level.

As illustrated in the figure below, the lowest end-to-end processing time is achieved when using a compression level of 1 or 2 with a single CPU and up to 5 when using 8 or 16 CPUs.

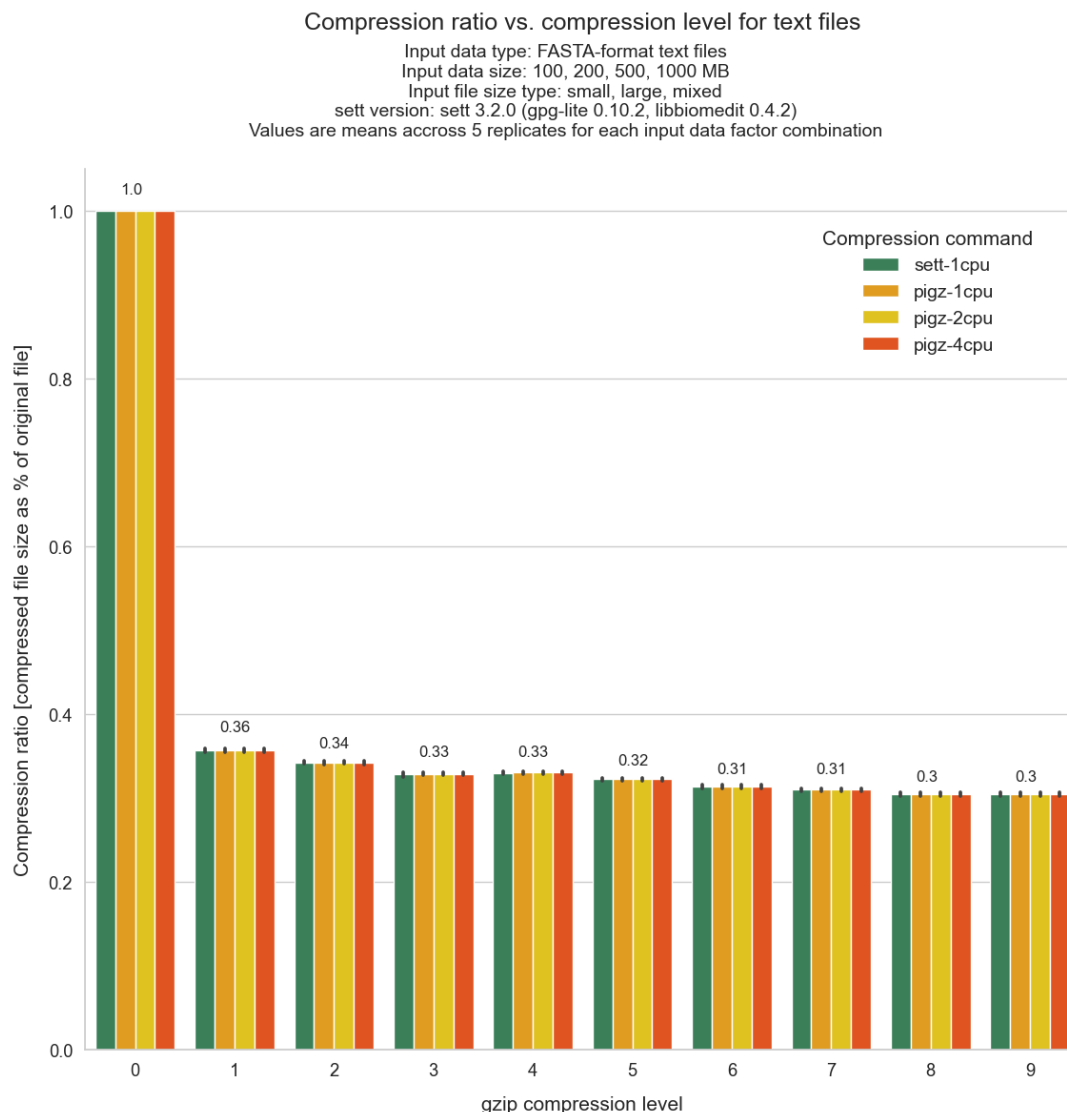


## Data size reduction achieved by different compression levels

The graph below shows how much data compression can be expected from each compression level when packaging text files with genomic data. Values give the size of the *sett* output file as a fraction of the input data size. For instance, the compression level 5 has an average compression ratio of 0.32, meaning that the size of the *sett* output is 32% of the input data size. In comparison, the compression level 0 has, obviously, a compression ratio of 1.0 since there is no data compression at all - the *sett* output has thus the same size as the input data.

All shown values are averages obtained across different input data sizes (ranging from 100 MB to 1 GB) and file size compositions (i.e. lots of small files vs a few large files). The error bars indicate confidence intervals - which are very small as there is little variation of compression ratio between data sizes and file size compositions.

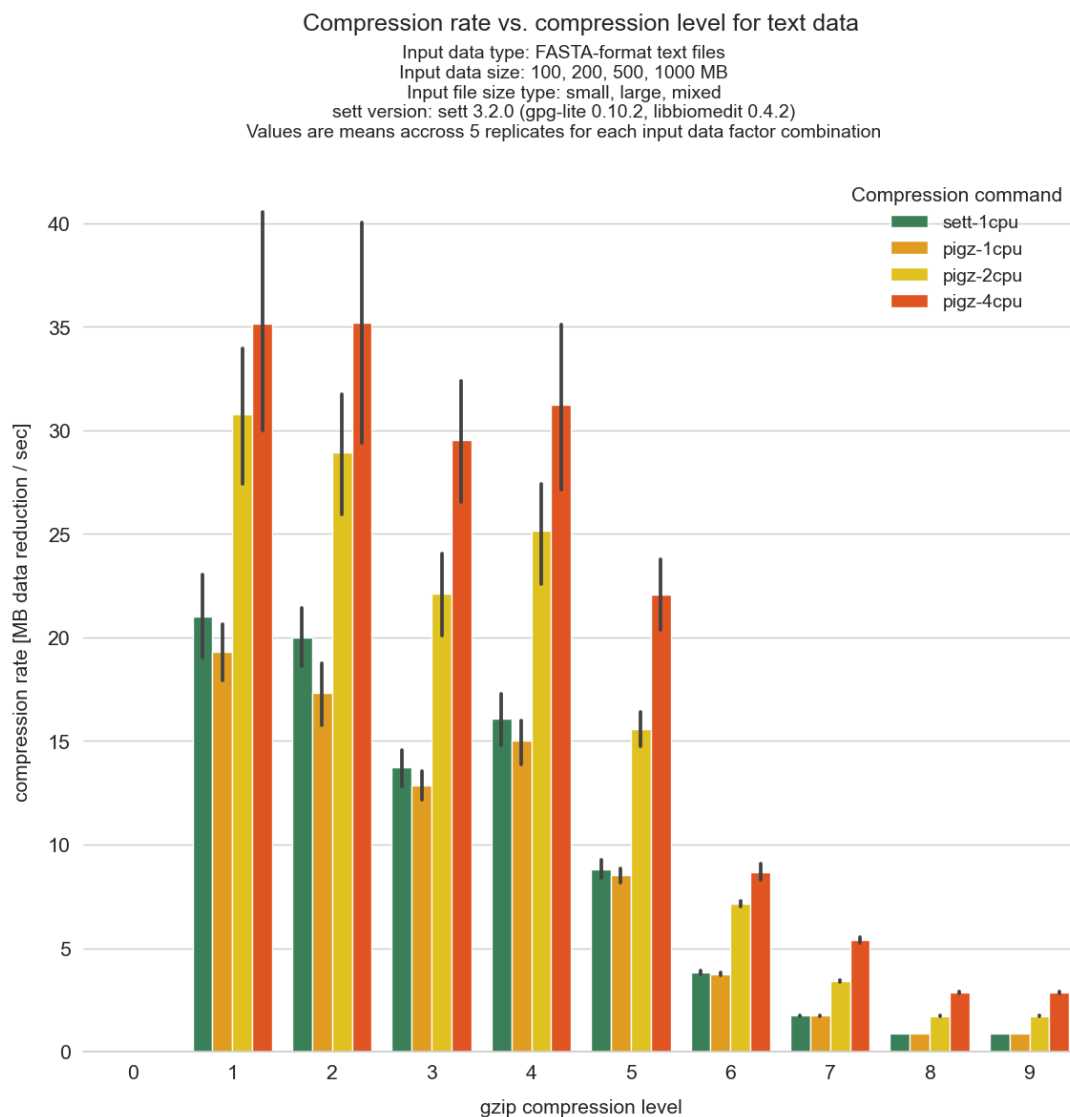
The graph also shows that, as expected, there are no differences in compression using *sett*'s internal compression method - the [python tarfile module](#) - or external compression with *gzip* (multi-threaded or not).



### Compression rate achieved by different compression levels

This graph shows the time efficiency of data compression (i.e. how much data reduction can be achieved per time unit) for the different compression levels. Lowering the compression level generally results in an increase in time efficiency - as increasing amounts of time must be spent to gain little additional compression. As can be seen, the most time efficient compression levels are levels 1 to 4.

Unsurprisingly, using multi-threaded compression (e.g. 8 or 16 CPUs) increases the rate of data compression, especially as compression level increases: the difference ratio between using 8 or 16 CPUs vs a single CPU is larger for higher compression levels.





## FREQUENTLY ASKED QUESTIONS AND BUG REPORTS

### 11.1 Bug reports

To report a problem with `sett` or if you have a question that is **not covered in the present documentation** please open an issue on our public gitlab repo <https://gitlab.com/biomedit/sett/-/issues>

### 11.2 Frequently asked questions

#### 11.2.1 Warning “Config: Unexpected keys: ...” at startup

- **Problem:** when invoking `sett` or `sett-cli`, a warning appears:

```
Config: Unexpected keys: ...
```

- **Solution:**
  - via GUI: Visit **Settings** and press *Save to config file*.
  - Via terminal: Edit your `config.json` (see *sett configuration file*) and remove the line defining the unexpected variable.

#### 11.2.2 Nothing happens when running `sett-gui` (Windows)

- **Problem:** when staring `sett-gui` in the terminal, nothing happens and the application does not start. In principle, this happens mostly on windows machines (on Linux/macOS, error messages should be displayed in the terminal).
- **Solution:** this generally indicates that an error occurred while the `sett-gui` application was launched. On Windows machines, this can be tricky to debug since the actual error is not displayed in the terminal. However, this problem often stems from a missing dependency, and you should therefore make sure that:
  - GnuPG is installed on your machine - *see here*.
  - `sett` was installed with all its python dependencies (this should in principle be the case if `sett` was installed using `pip`).

If this does not resolve the issue, you can try to manually start `sett-gui` in an interactive python3 session and see what the error message is:

1. Start a python3 session on your machine.
2. Run the following 2 commands, and see which error you get:

```
from sett.gui.__main__ import run
run()
```

### 11.2.3 Error “No module named ‘PySide2’” on sett startup (Linux / MacOS)

- **Problem:** when starting *sett*, the following error is displayed in the terminal: `ModuleNotFoundError: No module named 'PySide2'`. In addition, the following error (or something similar) might also be shown: `ImportError: libOpenGL.so.0: cannot open shared object file: No such file or directory` or the same error but with `libGLX.so.0` or `libEGL.so.1` instead of `libOpenGL.so.0`.
- **Solution:** depending on the operating system you are using, this error can have multiple causes.
  - **If using a recent operating system that fully supports PySide6** If your operating system is reasonably recent and supports PySide6, then the underlying problem is not the lack of PySide2, but most likely the fact that PySide6 is not correctly installed or is missing a dependency.

To fix the issue, please verify that:

- \* PySide6 is installed on your system: `pip list` will list all installed Python packages.
- \* The OpenGL library is installed on your system. OpenGL is an operating system library (not a Python module!) required by PySide6. However, this library is not always automatically installed when PySide6 is installed, and must in some cases be installed manually.

Here are commands for popular Linux distributions:

- **Linux Ubuntu or Debian:** `apt install libopengl0 libegl1`
- **Linux Fedora:** `dnf install libglvnd-opengl libglvnd-glx libglvnd-egl`
- **If using an older operating system that does not support PySide6** If you are using an older operating system (e.g. CentOS 7), then it is possible that PySide6 is not supported. In this case, *sett* must be run in so-called “legacy” mode, which requires PySide2 to be present on your system.

To fix the issue, please verify that:

- \* PySide2 is installed on your system.
- \* The “legacy” version of *sett* is installed on your system. The *sett* legacy version can be installed with: `pip install sett[legacy]`.

### 11.2.4 sett generates SHA-1 keys

- **Problem:** you are using *sett* and notice that it generates **SHA-1** keys. This is a problem because **SHA-1** is considered insecure and should NOT be used anymore.
- **Solution:** to generate non-**SHA-1** keys, you must use **GnuPG** command line executable. There is currently no option in *sett* to enforce the generation of non-**SHA-1** keys. It is possible to generate non-**SHA-1** keys by using following options (see [this](#) thread as well): `--cert-digest-algo SHA256 --s2k-digest-algo SHA256`. Instead of `SHA256`, select your preferred digest algorithm and/or one which is supported by your version of **GnuPG**, and compatible with other systems.

For more information on key management using **GnuPG** command line executable, kindly refer to *PGP key management with sett*.

## SOURCE CODE

**sett** is licensed under the GPLv3 (GNU General Public License) and the source code is available at <https://gitlab.com/biomedit/sett/>.

**sett** is developed as part of the [BioMedIT](#) project.