Gimmecert Documentation

Release 0.0-dev

Branko Majic

May 16, 2018

Contents

Gimmecert is a simple CLI tool for quickly issuing X.509 server and client certificates using locally-generated CA hierarchy with minimal hassle.

CHAPTER 1

Contents

1.1 About Gimmecert

Gimmecert is a simple CLI tool for quickly issuing X.509 server and client certificates using locally-generated CA hierarchy with minimal hassle.

The tool is useful for issuing certificates in:

- Local environment, when trying out a piece of new software that depends on use of certificates.
- Development environment, when it is necessary to issue certificates either for purpose of integration with other systems, or for ability to develop new feature that involves use of certificates.
- Testing/CI/CD environment, when it is necessary to deploy/configure tests to use certificates in order to ensure the tests are run properly and in full.

At time of this writing, Gimmecert is compatible with the following Python versions:

- *Python 3.4*
- *Python 3.5*
- *Python 3.6*

1.1.1 Why was this tool created?

The tool was created to remove the pain of setting-up a CA hierarchy, and then using this hierarchy to issue a couple of test certificates.

While there are existing tools that can be used to this end (in particular the OpenSSL's CLI and GnuTLS' `certtool`), the process of using them is tedious, slow, and error-prone.

There are some more long-lived solutions out there, in form of full-blown CAs, but those can be both an overkill and resource hog when all a person needs is a couple of certificates that can be thrown away.

1.1.2 Features

Gimmecert provides the following features:

- It is very easy to use. Commands are intuitive, and require minimal input from the user.
- Initialisation of CA hierarchy for issuing certificates. CA hierarchy depth can be specified, letting you easily simulate your production environment.
- Issuance of TLS server certificates, with any number of DNS subject alternative names.
- Issuance of TLS client certificates.
- All generated artifacts stored within a single sub-directory (`.gimmecert`), relative to directory where command is invoked. This allows you to easily issue per-project testing certificates.

1.1.3 Support

In case of problems with the tool, please do not hesitate to contact the author at **gimmecert (at) majic.rs**. Known issues and planned features are tracked on website:

- https://projects.majic.rs/gimmecert/

The tool is hosted on author's own server, alongside a mirror on Github:

- https://code.majic.rs/gimmecert
- https://github.com/azaghal/gimmecert

Documentation is available on:

- https://gimmecert.readthedocs.io/

1.1.4 License

1.2 Installation

Gimmecert can be easily installed using `pip`. Before installing it, make sure the following requirements have been met:

- You are running *Python 3.4+*.

In order to install latest stable release of *Gimmecert* using *pip*, run the following command:

```
pip install gimmecert
```

In order to install the latest development version of Gimmecert from git repository, use one of the following commands:

```
pip install -e git+https://code.majic.rs/gimmecert#egg=gimmecert
pip install -e git+https://github.com/azaghal/gimmecert#egg=gimmecert
```

1.3 Usage

Gimmecert provides a simple and clean CLI interface for all actions.

Description of all available commands, manadatory arguments, and optional arguments is also made available via CLI by either using the help flag (`-h` / `--help`) or help command.

Running the tool without any command or argument will output short usage instructions listing available commands.

To get more detailed general help on tool usage, and list of available commands with short description on what they do, run one of the following:

```
gimmecert -h
gimmecert --help
gimmecert help
```

To get more details on specific command, along what mandatory and positional arguments are available, simply provide the help flag when running them. For example:

```
gimmecert init -h
gimmecert init --help
```

1.3.1 Quickstart

Gimmecert stores all of its artefacts within the `.gimmecert` sub-directory (relative to where the command is run).

Start off by switching to your project directory:

```
cd ~/myproject/
```

Initialise the necessary directories and CA hierarchy:

```
gimmecert init
```

This will create a single CA, providing the following artifacts:

- `.gimmecert/ca/level1.key.pem` (private key)
- `.gimmecert/ca/level1.cert.pem` (certificate)
- `.gimmecert/ca/chain-full.cert.pem` (full CA chain, in this case same as `level1.cert.pem`)

Issue a server certificate:

```
gimmecert server myserver1
```

This will create the following artifacts for the server:

- `.gimmecert/server/myserver1.key.pem` (private key)
- `.gimmecert/server/myserver1.cert.pem` (certificate)

Resulting certificate will include its own name as one of the DNS subject alternative names.

Issue a client certificate:

```
gimmecert client myclient1
```

This will create the following artifacts for the client:

- `.gimmecert/client/myclient1.key.pem` (private key)
- `.gimmecert/client/myclient1.cert.pem` (certificate)

Issue a server certificate with additional DNS subject alternative names:

```
gimmecert server myserver2 myserver2.local service.example.com
```

This will create the following artifacts for the server:

- `.gimmecert/server/myserver2.key.pem` (private key)
- `.gimmecert/server/myserver2.cert.pem` (certificate)

This time around, the `myserver2` certificate will include `myserver2`, `myserver2.local`, and `service.example.com` as DNS subject alternative names.

Issue a server certificate by passing-in certificate signing request (CSR) from which the public key should be extracted:

```
openssl req -new -newkey rsa:2048 -nodes -keyout "/tmp/myserver3.key.pem" -subj "/
↪CN=ignoredname" -out "/tmp/myserver3.csr.pem"
gimmecert server --csr /tmp/myserver3.csr.pem myserver3
```

This will create the following artifacts for the server:

- `.gimmecert/server/myserver3.csr.pem` (CSR)
- `.gimmecert/server/myserver3.cert.pem` (certificate)

Renew existing certificates, keeping the same private key and naming:

```
gimmecert renew server myserver1
gimmecert renew server myclient1
```

Show information about CA hierarchy and issued certificates:

```
gimmecert status
```

1.3.2 Initialisation

Initialisation has to be run one before being being able to issue server and client certificates. This is done with:

```
gimmecert init
```

Initialisation will:

- Set-up a local directory.
- Initialise the CA hierarchy used for issuing server and client certificates. This includes creation of CA private keys (RSA 2048), as well as issuance of corresponding certificates.

If you attempt to run initialisation from the same directory twice, Gimmecert will refuse to do so. Should you need to recreate the hierarchy, simply remove the `.gimmcert/` directory, and start over. Keep in mind you will need to throw away all of generated key material and certificates.

The following directories are created as part of initialisation process:

- `.gimmecert/`, base directory.
- `.gimmecert/ca/`, used for storing CA private keys and certificates.
- `.gimmecert/server/`, used for storing server private keys and certificates.
- `.gimmecert/client/`, used for storing client private keys and certificates.

Both CA private keys and certificates are stored as OpenSSL-style PEM files. The naming convention for keys is `levelN.key.pem`, while for certificates it is `levelN.cert.pem`. `N` corresponds to CA level. Level 1 is the root/self-signed CA, level 2 is CA signed by level 1 CA and so forth.

In addition to individual CA certificates, Gimmecert will also store the full certificate chain (including the level 1 CA certificate) in file `chain-full.cert.pem`.

Subject DN naming convention for all CAs is `CN=BASENAME Level N CA`. `N` is the CA level, while `BASENAME` is by default equal to current (working) directory name.

By defualt the tool will initialise a one-level CA hierarchy (i.e. just the root CA).

Both the base name and CA hierarchy depth can be easily overridden by providing options (both long and short forms are available):

```
gimmecert init --ca-base-name "My Project" --ca-hierarchy-depth 3
gimmecert init -b "My Project" -d 3
```

The above examples would both result in creation of the following CA artifacts:

- `.gimmecert/ca/level1.key.pem`
- `.gimmecert/ca/level1.cert.pem` (subject DN `My Project Level 1 CA`)
- `.gimmecert/ca/level2.key.pem`
- `.gimmecert/ca/level2.cert.pem` (subject DN `My Project Level 2 CA`)
- `.gimmecert/ca/level3.key.pem`
- `.gimmecert/ca/level3.cert.pem` (subject DN `My Project Level 3 CA`)
- `.gimmecert/ca/chain-full.cert.pem`

1.3.3 Issuing server certificates

Server certificates can be issued once the initialisation is complete. Command supports passing-in additional DNS subject alternative names as additional positional arguments:

```
gimmecert server NAME [DNS_NAME [DNS_NAME ...]]
```

The command will:

- Generate a 2048-bit RSA private key.
- Issue a certificate associated with the generated private key using the leaf CA (the one deepest in hierachy).

Resulting private keys and certificates are stored within directory `.gimmecert/server/`. Private key naming convention is `NAME.key.pem`, while certificates are stored as `NAME.cert.pem`. In both cases the OpenSSL-style PEM format is used for storage.

Subject DN naming convention for server certificates is `CN=NAME`, where `NAME` is passed-in via positional argument.

By default the certificate will include the passed-in server name as one of its DNS subject alternative names, but additional DNS names can be passed-in as well. For example:

```
gimmecert server myserver myserver.local service.example.com
```

Key usage and extended key usage in certificate are set typical TLS server use (e.g. *digital signature* + *key encipherment* for KU, and *TLS WWW server authentication* for EKU).

Rerunning the command will not overwrite existing data.

Note: For changing the list of additional subject alternative names included in already issued server certificates, see the `--update-dns-names` option in the `gimmecert renew` command.

In addition to generating a private key, it is also possible to pass-in a certificate signing request (CSR). If specified path is a dash (-), CSR is read from standard input. The resulting certificate will contain public key from the CSR. All other information stored in the CSR (naming, extensions) is ignored. For example:

```
# Issue server certificate by passing-in path to a generated CSR.
gimmecert server --csr /tmp/myown.csr.pem myserver

# Issue server certificate by reading the CSR from standard input.
gimmecert server --csr - myserver

# Issue server certificate by reading the CSR from standard input,
# using redirection.
gimmecert server --csr - myserver < /tmp/myown.csr.pem
```

The passed-in CSR will be stored alongside certificate, under `.gimmecert/server/NAME.csr.pem`.

1.3.4 Issuing client certificates

Client certificates can be issued once the initialisation is complete. Command accepts a single positional argument:

```
gimmecert client NAME
```

The command will:

- Generate a 2048-bit RSA private key.
- Issue a certificate associated with the generated private key using the leaf CA (the one deepest in hierachy).

Rerunning the command will not overwrite existing data.

Resulting private keys and certificates are stored within directory `.gimmecert/client/`. Private key naming convention is `NAME.key.pem`, while certificates are stored as `NAME.cert.pem`. In both cases the OpenSSL-style PEM format is used for storage.

Subject DN naming convention for client certificates is `CN=NAME`, where `NAME` is passed-in via positional argument.

Key usage and extended key usage in certificate are set typical TLS client use (e.g. *digital signature* + *key encipherment* for KU, and *TLS WWW client authentication* for EKU).

In addition to generating a private key, it is also possible to pass-in a certificate signing request (CSR). If specified path is a dash (-), CSR is read from standard input. The resulting certificate will contain public key from the CSR. All other information stored in the CSR (naming, extensions) is ignored. For example:

```
# Issue client certificate by passing-in path to a generated CSR.
gimmecert client --csr /tmp/myown.csr.pem myclient

```

(continues on next page)

(continued from previous page)

```
# Issue client certificate by reading the CSR from standard input.
gimmecert client --csr - myclient

# Issue client certificate by reading the CSR from standard input,
# using redirection.
gimmecert client --csr - myclient < /tmp/myown.csr.pem
```

The passed-in CSR will be stored alongside certificate, under `.gimmecert/client/NAME.csr.pem`.

1.3.5 Renewing certificates

Both client and server certificates can be renewed by simply providing the type and name. This is useful when a certificate has expired, and it should be renewed with identical naming and private key. Command requires two positional argumensts:

```
gimmecert renew (server|client) NAME
```

The command will:

- By default keep the existing private key generated for end entity (new one can be requested as well).
- Re-use naming and any extensions stored in existing certificate.
- Overwrite the existing certificate with a new one.
- Show information where the artifacts can be grabbed from.

To also generate a new private key during renewal, use the `--new-private-key` or `-p` option. For example:

```
gimmecert renew --new-private-key server myserver
gimmecert renew -p server my server
```

To replace the existing private key or CSR during renewal with a new CSR, use the `--csr` option and pass along path to the file. If specified path is a dash (`-`), CSR is read from standard input. For example:

```
gimmecert renew --csr /tmp/myserver.csr.pem server myserver
gimmecert renew --csr - server myserver < /tmp/myserver.csr.pem
gimmecert renew --csr - client myclient
```

If you initially made a mistake when providing additional DNS subject alternative names for a server certificate, you can easily fix this with the `--update-dns-names` or `-u` option:

```
# Replace existing additional names with just one name.
gimmecert renew server --update-dns-names "correctname.example.com" myserver

# Replace existing additional names with mutliple names.
gimmecert renew server --update-dns-names "correctname1.example.com,correctname2.
↪example.com" myserver

# Remove additional names altogether.
gimmecert renew server --update-dns-names "" myserver
```

1.3.6 Getting information about CA hierarchy and issued certificates

In order to show information about the CA hierarchy and issued certificates simply run the status command:

```
gimmecert status
```

The command will:

- Show information about every CA in generated hierarchy (subject DN, validity, certificate paths, whether the CA is used for issuing end entity certificates).
- Show information about all issued server certificates (subject DN, DNS subject alternative names, validity, private key or CSR path, certificate path).
- Show information about all issued client certificates (subject DN, validity, private key or CSR path, certificate path).

Validity of all certificates is shown in UTC.

Command can also be used for checking if Gimmecert has been initialised in local directory or not.

1.4 Development

This section provides information on development process for the project, including instructions on how to set-up a development environment or run the tests locally.

1.4.1 Preparing development environment

There is a number of different ways a development environment can be set-up. The process outlined here is centered around virtualenvwrapper. Instructions have been tailored for a GNU/Linux system.

Before proceeding, ensure you have the following system-wide packages installed:

- Python, version 3.4+.
- virtualenvwrapper.

With those in place, do the following:

1. Clone the git repository:

   ```
   git clone https://code.majic.rs/gimmecert/
   ```

2. Change directory:

   ```
   cd gimmecert/
   ```

3. Create Python virtual environment:

   Warning: Make sure to specify Python 3 as interpreter.

   ```
   mkvirtualenv -a . -p /usr/bin/python3 gimmecert
   ```

4. Install development requirements:

   ```
   pip install -e .[devel]
   ```

5. At this point, any time you want to run tests etc, you can easily switch to correct environment (this will also put you in project root directory) with:

```
workon gimmecert
```

1.4.2 Testing

Project includes both unit tests (within `tests/` directory) , and functional tests (within `functional_tests/` directory).

Tests can be run in a number of different ways, depending on what you want to test and how.

To run the unit tests via setup script, run the following command from repository root:

```
python setup.py test
```

To run the unit tests directly, run the following command from repository root:

```
pytest
```

Tests can be also run with coverage checks. By default coverage is configured to report to standard output. Report will only list files which lack coverage. For each file a percentage will be shown, as well as line numbers that were not covered by tests. To include coverage checks, run tests with:

```
pytest --cov
```

Warning: Gimmecert project has 100% coverage requirement. Anything below will trigger failures when coverage checks are run.

Should you desire to generate coverage report in HTML format, run (coverage report will be put into directory `coverage/`):

```
pytest --cov --cov-report=html:coverage/
```

Functional tests must be run explicitly (since they tend to take more time), with:

```
pytest functional_tests/
```

In addition to proper linting, implemented code should be pruned of unused imports and variables. Linting should be conformant to PEP8, with the exception of line length, which is allowed to be up to 160 characters. Linting and code sanity checks are not executed automatically, but can easily be run with:

```
flake8
```

Documentation should be buildable at all times. Documentation build can be triggered with a simple:

```
cd docs/
make html
```

Tests can also be run using tox:

Note: When running tests via `tox`, functional tests and coverage checks are included as well.

```
# Run full suite of tests on all supported Python versions.
tox

# List available tox environments.
tox -l

# Run tests against specific Python version.
tox -e py35

# Run documentation and linting tests only.
tox -e doc,lint
```

1.4.3 Building documentation

Documentation is written in reStructuredText and built via Sphinx.

To build documentation, run:

```
cd docs/
make html
```

Resulting documentation will be stored in HTML format in directory `docs/_build/html/`.

1.4.4 Versioning schema

Project employs semantic versioning schema. In short:

- Each version is composed of major, minor, and patch number. For example, in version `1.2.3`, `1` is the major, `2` is the minor, and `3` is the patch number.
- Major number is bumped when making a backwards incompatible change.
- Minor number is bumped when new features or changes are made without breaking backwards compatibility.
- Patch number is bumped when backporting bug or security fixes into an older release.

In addition to versioning schema, project employs a specific nomenclature for naming the branches:

- All new development (both for features and bug/security fixes) uses master branch as the base.
- Features and bug/security fixes are implemented in a local branch based on the master branch. Local branches are named after the lower-cased issue number. For example, if the issuer number is `GC-43`, the implementation branch will be named `gc-43`. Normally these branches are only local, but if necessary they can be pushed to central repository for collaboration or preview purposes.
- Patch releases are based off the maintenance branches. Mainteance branches are named after the `MAJOR` and `MINOR` number of the version - `maintenance/MAJOR.MINOR`. For example, if a new release is made with version `1.2.0`, the corresponding branch that is created for maintenance will be named `maintenance/1.2` (notice the absence of `.0` at the end).

1.4.5 Backporting fixes

From time to time it might become useful to apply a bug/security fix to both the master branch, and to maintenace branch.

When a bug should be applied to maintenance branch as well, procedure is as follows:

1. Create a new bug report in issue tracker. Target version should be either the next minor or next major release (i.e. whatver will get released from the master branch).
2. Create a copy of the bug report, modifying the issue title to include phrase `(backport to MAJOR.MINOR)` at the end, with `MAJOR` and `MINOR` replaced with correct versioning information for the maintenance branch. Make sure to set correct target version (patch release).
3. Resolve the bug for next major/minor release.
4. Resolve the bug in maintenace branch by backporting (cherry-picking if possible) the fix into maintenace branch. Make sure to resign (cherry-picking invalidates OpenPGP signature) and reword (to reference the backport issue) the commit.

1.4.6 Release notes

Release notes are written in parallel to resolving project issues, in the `docs/releasenotes.rst` file. In other words, any time a new feature, bug fix etc is implemented, an entry should be created in the relase notes file. This applies for tasks and user stories as well.

By ensuring the release notes are always up-to-date, the release process is simpler, faster, and less error prone.

Release notes are always added under section title **NEXT RELEASE**. This placeholder section title is replaced during the release process.

Release notes for each version consist out of two parts - the general release description, and listing of resolved issues.

General description provides a high-level overview of new functionality and fixes included in the release, and points to any important/breaking changes.

The listing of resolved issues is split-up based on issue type, and lists all issues that have been resolved in the given release. Each issue in the list is provied as URL link pointing to issue URL in the issue tracker, with the link text in format `ISSUE_NUMBER: ISSUE_TITLE`. Both issue number and issue title are taken from the issue tracker.

To provide a more visual example, template for single release note is as follows:

```
NEXT RELEASE
------------

[General description of release.]

Resolved issues:

- **User stories**:

  - `ISSUER_NUMBER: ISSUE_TITLE <ISSUE_URL>`_
  - `ISSUER_NUMBER: ISSUE_TITLE <ISSUE_URL>`_
- **Feature requests**:

  - `ISSUER_NUMBER: ISSUE_TITLE <ISSUE_URL>`_
  - `ISSUER_NUMBER: ISSUE_TITLE <ISSUE_URL>`_
- **Enhancements**:

  - `ISSUER_NUMBER: ISSUE_TITLE <ISSUE_URL>`_
  - `ISSUER_NUMBER: ISSUE_TITLE <ISSUE_URL>`_
- **Tasks**:

  - `ISSUER_NUMBER: ISSUE_TITLE <ISSUE_URL>`_
  - `ISSUER_NUMBER: ISSUE_TITLE <ISSUE_URL>`_
```

1.4.7 Release process

The release process for Gimmecert is centered around the use of included `release.sh` script. The script takes care of a number of tedious tasks, including:

- Updating the version information in release notes and `setup.py`.
- Tagging a release.
- Starting a new section in release notes.
- Switching version back to development in `setup.py`.
- Publishing changes to origin Git repository and publishing release to PyPI.

When releasing a new major/minor version (from the master branch), the release script will take care of setting-up a maintenance branch as well.

Patch releases should be done from maintenance branches. New major/minor releases should be done from the master branch.

Warning: Keep in mind that the release script is interactive, it cannot be run unattended.

Perform the following steps in order to release new version of Gimmecert:

1. Make sure that Git is correctly set-up for signing using GnuPG, and that the necessary key material is available.
2. Verify that there are no outstanding issues for this release in the issue tracker.
3. Switch to project virtual environment.
4. Ensure that the repository is synchronised with origin, and that a correct branch is checked out (master or maintenance).
5. Go through release notes for `NEXT VERSION`, and ensure the general description and listed issues look fine. Make any necessary changes, commit them, and push them to the origin.
6. Prepare the release:

   Warning: Make sure to provide correct version.

   ```
   ./release.sh prepare VERSION
   ```

7. Verify that the preparation process was successful.
8. Publish the release:

   Warning: Make sure to provide correct version.

   ```
   ./release.sh publish VERSION
   ```

9. Build release documentation on Read the Docs project page, and update the default version if this was a new major/minor release.
10. Verify documentation looks good on Read the Docs documentation page.
11. Mark the release issue as resolved in the issue tracker.

12. Release the version via release center in the issue tracker. Upload source archive from the `dist/` directory.
13. Archive outdated releases in the issue tracker.

1.5 Release notes

1.5.1 0.1.0

First release of Gimmecert. Implements ability to set-up per-directory CA hierarchy that can then be used to issue server and client certificates.

Resolved issues:

- **User stories**:
  - GC-4: As a system integrator, I want to easily issue server and client certificates so that I can quickly test software that requires them
  - GC-5: As a system integrator, I want to initialise a CA hierarchy in project directory in order to use it within the project
  - GC-6: As a system integrator, I want to issue server certificates so I can deploy them for use with server applications I use
  - GC-7: As a system integrator, I want to issue client certificates so I can deploy them for use with client applications I use
  - GC-8: As a system integrator, I want to get status of current CA hierarchy and issued certificates so I could determine if I need to take an action
  - GC-9: As a system integrator, I want to renew server or client certificate in order to change the additional naming or renew expigration date
  - GC-10: As a system integrator, I want to be able to see tool's help in CLI so I can remind myself what commands are available
  - GC-21: As a system integrator, I want to be able to issue certificates using a CSR so I can generate my own private key
- **Feature requests**:
  - GC-2: Project skeleton
  - GC-3: Ability to initialise CA hierarchy
  - GC-11: Initial skeleton CLI implementation
  - GC-12: Initial installation and usage instructions
  - GC-15: Ability to issue server certificates
  - GC-16: Ability to issue client certificates
  - GC-19: Ability to update server certificate DNS subject alternative names
  - GC-18: Ability to renew existing certificates
  - GC-20: Ability to display status
  - GC-22: Ability to provide CSR for issuing and renewing certificates
- **Enhancements**:
  - GC-14: Clean-up test runtime configuration and imrpove usability

- **Tasks**:
  - GC-1: Set-up project infrastructure
  - GC-17: Refactor CLI command handling and relevant tests

CHAPTER 2

Indices and tables

- genindex
- modindex
- search