Let's Encrypt Documentation

Release 0.1.0.dev0

Let's Encrypt Project

October 26, 2015

Contents

CHAPTER 1

Introduction

1.1 Disclaimer

This is a **DEVELOPER PREVIEW** intended for developers and testers only.

DO NOT RUN THIS CODE ON A PRODUCTION SERVER. IT WILL INSTALL CERTIFICATES SIGNED BY A TEST CA, AND WILL CAUSE CERT WARNINGS FOR USERS.

Browser-trusted certificates will be available in the coming months.

For more information regarding the status of the project, please see https://letsencrypt.org. Be sure to checkout the Frequently Asked Questions (FAQ).

1.2 About the Let's Encrypt Client

In short: getting and installing SSL/TLS certificates made easy (watch demo video).

The Let's Encrypt Client is a tool to automatically receive and install X.509 certificates to enable TLS on servers. The client will interoperate with the Let's Encrypt CA which will be issuing browser-trusted certificates for free.

It's all automated:

- The tool will prove domain control to the CA and submit a CSR (Certificate Signing Request).
- If domain control has been proven, a certificate will get issued and the tool will automatically install it.

All you need to do to sign a single domain is:

```
user@www:~$ sudo letsencrypt -d www.example.org auth
```

For multiple domains (SAN) use:

```
user@www:~$ sudo letsencrypt -d www.example.org -d example.org auth
```

and if you have a compatible web server (Apache or Nginx), Let's Encrypt can not only get a new certificate, but also deploy it and configure your server automatically!:

```
user@www:~$ sudo letsencrypt -d www.example.org run
```

Encrypt ALL the things!

1.2.1 Current Features

- Supports multiple web servers:
  - apache/2.x (tested and working on Ubuntu Linux)
  - nginx/0.8.48+ (under development)
  - standalone (runs its own simple webserver to prove you control a domain)
- The private key is generated locally on your system.
- Can talk to the Let's Encrypt (demo) CA or optionally to other ACME compliant services.
- Can get domain-validated (DV) certificates.
- Can revoke certificates.
- Adjustable RSA key bit-length (2048 (default), 4096, ...).
- Can optionally install a http -> https redirect, so your site effectively runs https only (Apache only)
- Fully automated.
- Configuration changes are logged and can be reverted.
- Text and ncurses UI.
- Free and Open Source Software, made with Python.

1.2.2 Installation Instructions

Official **documentation**, including installation instructions, is available at https://letsencrypt.readthedocs.org.

1.2.3 Links

Documentation: https://letsencrypt.readthedocs.org

Software project: https://github.com/letsencrypt/letsencrypt

Notes for developers: CONTRIBUTING.md

Main Website: https://letsencrypt.org/

IRC Channel: #letsencrypt on Freenode

Community: https://community.letsencrypt.org

Mailing list: client-dev (to subscribe without a Google account, send an email to client-dev+subscribe@letsencrypt.org)

1.3 ChangeLog

Please note: the change log will only get updated after first release - for now please use the commit log.

1.3.1 Release 0.1.0 (not released yet)

New Features:

- ...

Fixes:

- ...

Other changes:

- ...

1.3.2 Release 0.0.0 (not released yet)

Initial release.

CHAPTER 2

User Guide

Table of Contents

2.1 Installation

2.1.1 letsencrypt-auto

`letsencrypt-auto` is a wrapper which installs some dependencies from your OS standard package repositories (e.g using `apt-get` or `yum`), and for other dependencies it sets up a virtualized Python environment with packages downloaded from PyPI [1]. It also provides automated updates.

Firstly, please install Git and run the following commands:

```
git clone https://github.com/letsencrypt/letsencrypt
cd letsencrypt
```

Warning: Alternatively you could download the ZIP archive and extract the snapshot of our repository, but it's strongly recommended to use the above method instead.

To install and run the client you just need to type:

```
./letsencrypt-auto
```

Throughout the documentation, whenever you see references to `letsencrypt` script/binary, you can substitute in `letsencrypt-auto`. For example, to get the help you would type:

[1] By using this virtualized Python environment (virtualenv) we don't pollute the main OS space with packages from PyPI!

```
./letsencrypt-auto --help
```

2.1.2 Running with Docker

Docker is an amazingly simple and quick way to obtain a certificate. However, this mode of operation is unable to install certificates or configure your webserver, because our installer plugins cannot reach it from inside the Docker container.

You should definitely read the *Where are my certificates?* section, in order to know how to manage the certs manually. https://github.com/letsencrypt/letsencrypt/wiki/Ciphersuite-guidance provides some information about recommended ciphersuites. If none of these make much sense to you, you should definitely use the *letsencrypt-auto* method, which enables you to use installer plugins that cover both of those hard topics.

If you're still not convinced and have decided to use this method, from the server that the domain your requesting a cert for resolves to, install Docker, issue the following command:

```
sudo docker run -it --rm -p 443:443 --name letsencrypt \
            -v "/etc/letsencrypt:/etc/letsencrypt" \
            -v "/var/lib/letsencrypt:/var/lib/letsencrypt" \
            quay.io/letsencrypt/letsencrypt:latest auth
```

and follow the instructions (note that `auth` command is explicitly used - no installer plugins involved). Your new cert will be available in `/etc/letsencrypt/live` on the host.

2.1.3 Distro packages

Unfortunately, this is an ongoing effort. If you'd like to package Let's Encrypt client for your distribution of choice please have a look at the Packaging Guide.

2.1.4 From source

Installation from source is only supported for developers and the whole process is described in the Developer Guide.

Warning: Please do **not** use `python setup.py install` or `python pip install .`. Please do **not** attempt the installation commands as superuser/root and/or without virtual environment, e.g. `sudo python setup.py install`, `sudo pip install`, `sudo ./venv/bin/...`. These modes of operation might corrupt your operating system and are **not supported** by the Let's Encrypt team!

2.1.5 Comparison of different methods

Unless you have a very specific requirements, we kindly ask you to use the *letsencrypt-auto* method. It's the fastest, the most thoroughly tested and the most reliable way of getting our software and the free SSL certificates!

2.2 Plugins

Officially supported plugins:

Plugin	A	I	Notes and status
standalone	Y	N	Very stable. Uses port 80 (force by `--standalone-supported-challenges simpleHttp`) or 443 (force by `standalone-supported-challenges dvsni`).
apache	Y	Y	Alpha. Automates Apache installation, works fairly well but on Debian-based distributions only for now.
webroot	Y	N	Works with already running webserver, by writing necessary files to the disk (`--webroot-path` should be pointed to your `public_html`). Currently, when multiple domains are specified (`-d`), they must all use the same web root path.
manual	Y	N	Hidden from standard UI, use with `--a manual`. Requires to copy and paste commands into a new terminal session. Allows to run client on machine different than target webserver, e.g. your laptop.
nginx	Y	Y	Very experimental. Not included in *letsencrypt-auto*.

Third party plugins are listed at https://github.com/letsencrypt/letsencrypt/wiki/Plugins. If that that's not enough, you can always *write your own plugin*.

2.3 Renewal

Note: Let's Encrypt CA issues short lived certificates (90 days). Make sure you renew the certificates at least once in 3 months.

In order to renew certificates simply call the `letsencrypt` (or *letsencrypt-auto*) again, and use the same values when prompted. You can automate it slightly by passing necessary flags on the CLI (see `--help all`), or even further using the *Configuration file*. If you're sure that UI doesn't prompt for any details you can add the command to `crontab` (make it less than every 90 days to avoid problems, say every month).

Let's Encrypt is working hard on automating the renewal process. Until the tool is ready, we are sorry for the inconvenience!

2.4 Where are my certificates?

First of all, we encourage you to use Apache or nginx installers, both which perform the certificate managemant automatically. If, however, you prefer to manage everything by hand, this section provides information on where to find necessary files.

All generated keys and issued certificates can be found in `/etc/letsencrypt/live/$domain`. Rather than copying, please point your (web) server configuration directly to those files (or create symlinks). During the *renewal*, `/etc/letsencrypt/live` is updated with the latest necessary files.

Note: `/etc/letsencrypt/archive` and `/etc/letsencrypt/keys` contain all previous keys and certificates, while `/etc/letsencrypt/live` symlinks to the latest versions.

The following files are available:

`privkey.pem` Private key for the certificate.

> **Warning:** This **must be kept secret at all times**! Never share it with anyone, including Let's Encrypt developers. You cannot put it into safe, however - your server still needs to access this file in order for SSL/TLS to work.

This is what Apache needs for SSLCertificateKeyFile, and nginx for ssl_certificate_key.

cert.pem Server certificate only.

This is what Apache needs for SSLCertificateFile.

chain.pem All certificates that need to be served by the browser **excluding** server certificate, i.e. root and intermediate certificates only.

This is what Apache needs for SSLCertificateChainFile.

fullchain.pem All certificates, **including** server certificate. This is concatenation of `chain.pem` and `cert.pem`.

This is what nginx needs for ssl_certificate.

For both chain files, all certificates are ordered from root (primary certificate) towards leaf.

Please note, that **you must use** either `chain.pem` or `fullchain.pem`. In case of webservers, using only `cert.pem`, will cause nasty errors served through the browsers!

Note: All files are PEM-encoded (as the filename suffix suggests). If you need other format, such as DER or PFX, then you could convert using `openssl`, but this means you will not benefit from automatic *renewal*!

2.5 Configuration file

It is possible to specify configuration file with `letsencrypt-auto --config cli.ini` (or shorter `-c cli.ini`). An example configuration file is shown below:

```
# This is an example of the kind of things you can do in a configuration file.
# All flags used by the client can be configured here. Run Let's Encrypt with
# "--help" to learn more about the available options.

# Use a 4096 bit RSA key instead of 2048
rsa-key-size = 4096

# Always use the staging/testing server
server = https://acme-staging.api.letsencrypt.org/directory

# Uncomment and update to register with the specified e-mail address
# email = foo@example.com

# Uncomment to use a text interface instead of ncurses
# text = True

# Uncomment to use the standalone authenticator on port 443
# authenticator = standalone
# standalone-supported-challenges = dvsni

# Uncomment to use the webroot authenticator. Replace webroot-path with the
# path to the public_html / webroot folder being served by your web server.
# authenticator = webroot
# webroot-path = /usr/share/nginx/html
```

By default, the following locations are searched:

- `/etc/letsencrypt/cli.ini`
- `$XDG_CONFIG_HOME/letsencrypt/cli.ini` (or `~/.config/letsencrypt/cli.ini` if `$XDG_CONFIG_HOME` is not set).

2.6 Getting help

If you're having problems you can chat with us on IRC (#letsencrypt @ Freenode) or get support on our forums.

If you find a bug in the software, please do report it in our issue tracker. Remember to give us us as much information as possible:

- copy and paste exact command line used and the output (though mind that the latter might include some personally identifiable information, including your email and domains)
- copy and paste logs from `/var/log/letsencrypt` (though mind they also might contain personally identifiable information)
- copy and paste `letsencrypt --version` output
- your operating system, including specific version
- specify which *installation* method you've chosen

CHAPTER 3

Developer Guide

Table of Contents

3.1 Hacking

3.1.1 Running a local copy of the client

Running the client in developer mode from your local tree is a little different than running `letsencrypt-auto`. To get set up, do these things once:

```
git clone https://github.com/letsencrypt/letsencrypt
cd letsencrypt
./bootstrap/install-deps.sh
./bootstrap/dev/venv.sh
```

Then in each shell where you're working on the client, do:

```
source ./venv/bin/activate
```

After that, your shell will be using the virtual environment, and you run the client by typing:

```
letsencrypt
```

Activating a shell in this way makes it easier to run unit tests with `tox` and integration tests, as described below. To reverse this, you can type `deactivate`. More information can be found in the virtualenv docs.

3.1.2 Find issues to work on

You can find the open issues in the github issue tracker. Comparatively easy ones are marked Good Volunteer Task. If you're starting work on something, post a comment to let others know and seek feedback on your plan where appropriate.

Once you've got a working branch, you can open a pull request. All changes in your pull request must have thorough unit test coverage, pass our *integration* tests, and be compliant with the *coding style*.

3.1.3 Testing

The following tools are there to help you:

- `tox` starts a full set of tests. Please make sure you run it before submitting a new pull request.
- `tox -e cover` checks the test coverage only. Calling the `./tox.cover.sh` script directly (or even `./tox.cover.sh $pkg1 $pkg2 ...` for any subpackages) might be a bit quicker, though.
- `tox -e lint` checks the style of the whole project, while `pylint --rcfile=.pylintrc path` will check a single file or specific directory only.
- For debugging, we recommend `pip install ipdb` and putting `import ipdb; ipdb.set_trace()` statement inside the source code. Alternatively, you can use Python's standard library `pdb`, but you won't get TAB completion...

Integration testing with the boulder CA

Generally it is sufficient to open a pull request and let Github and Travis run integration tests for you.

Mac OS X users: Run `/tests/mac-bootstrap.sh` instead of `boulder-start.sh` to install dependencies, configure the environment, and start boulder.

Otherwise, install Go 1.5, libtool-ltdl, mariadb-server and rabbitmq-server and then start Boulder, an ACME CA server:

```
./tests/boulder-start.sh
```

The script will download, compile and run the executable; please be patient - it will take some time... Once its ready, you will see `Server running, listening on 127.0.0.1:4000...`. Add an `/etc/hosts` entry pointing `le.wtf` to 127.0.0.1. You may now run (in a separate terminal):

```
./tests/boulder-integration.sh && echo OK || echo FAIL
```

If you would like to test `letsencrypt_nginx` plugin (highly encouraged) make sure to install prerequisites as listed in `letsencrypt-nginx/tests/boulder-integration.sh` and rerun the integration tests suite.

3.2 Code components and layout

acme contains all protocol specific code

letsencrypt all client code

3.2.1 Plugin-architecture

Let's Encrypt has a plugin architecture to facilitate support for different webservers, other TLS servers, and operating systems. The interfaces available for plugins to implement are defined in interfaces.py.

The most common kind of plugin is a "Configurator", which is likely to implement the *IAuthenticator* and *IInstaller* interfaces (though some Configurators may implement just one of those).

There are also *IDisplay* plugins, which implement bindings to alternative UI libraries.

3.2.2 Authenticators

Authenticators are plugins designed to prove that this client deserves a certificate for some domain name by solving challenges received from the ACME server. From the protocol, there are essentially two different types of challenges. Challenges that must be solved by individual plugins in order to satisfy domain validation (subclasses of `DVChallenge`, i.e. `DVSNI`, `SimpleHTTPS`, `DNS`) and continuity specific challenges (subclasses of `ContinuityChallenge`, i.e. `RecoveryToken`, `RecoveryContact`, `ProofOfPossession`). Continuity challenges are always handled by the *ContinuityAuthenticator*, while plugins are expected to handle `DVChallenge` types. Right now, we have two authenticator plugins, the `ApacheConfigurator` and the `StandaloneAuthenticator`. The Standalone and Apache authenticators only solve the `DVSNI` challenge currently. (You can set which challenges your authenticator can handle through the *get_chall_pref()*.

(FYI: We also have a partial implementation for a `DNSAuthenticator` in a separate branch).

3.2.3 Installer

Installers plugins exist to actually setup the certificate in a server, possibly tweak the security configuration to make it more correct and secure (Fix some mixed content problems, turn on HSTS, redirect to HTTPS, etc). Installer plugins tell the main client about their abilities to do the latter via the *supported_enhancements()* call. We currently have two Installers in the tree, the `ApacheConfigurator`. and the `NginxConfigurator`. External projects have made some progress toward support for IIS, Icecast and Plesk.

Installers and Authenticators will oftentimes be the same class/object (because for instance both tasks can be performed by a webserver like nginx) though this is not always the case (the standalone plugin is an authenticator that listens on port 443, but it cannot install certs; a postfix plugin would be an installer but not an authenticator).

Installers and Authenticators are kept separate because it should be possible to use the `StandaloneAuthenticator` (it sets up its own Python server to perform challenges) with a program that cannot solve challenges itself (Such as MTA installers).

3.2.4 Installer Development

There are a few existing classes that may be beneficial while developing a new *IInstaller*. Installers aimed to reconfigure UNIX servers may use Augeas for configuration parsing and can inherit from `AugeasConfigurator` class to handle much of the interface. Installers that are unable to use Augeas may still find the *Reverter* class helpful in handling configuration checkpoints and rollback.

Display

We currently offer a pythondialog and "text" mode for displays. Display plugins implement the *IDisplay* interface.

3.3 Writing your own plugin

Let's Encrypt client supports dynamic discovery of plugins through the setuptools entry points. This way you can, for example, create a custom implementation of *IAuthenticator* or the *IInstaller* without having to merge it with the core upstream source code. An example is provided in `examples/plugins/` directory.

Warning: Please be aware though that as this client is still in a developer-preview stage, the API may undergo a few changes. If you believe the plugin will be beneficial to the community, please consider submitting a pull request to the repo and we will update it with any necessary API changes.

3.4 Coding style

Please:

1. **Be consistent with the rest of the code**.
2. Read PEP 8 - Style Guide for Python Code.
3. Follow the Google Python Style Guide, with the exception that we use Sphinx-style documentation:

```
    def foo(arg):
        """Short description.

        :param int arg: Some number.

        :returns: Argument
        :rtype: int

        """
        return arg
```

4. Remember to use `pylint`.

3.5 Submitting a pull request

Steps:

1. Write your code!

2. Make sure your environment is set up properly and that you're in your virtualenv. You can do this by running `./bootstrap/dev/venv.sh`. (this is a **very important** step)
3. Run `./pep8.travis.sh` to do a cursory check of your code style. Fix any errors.
4. Run `tox -e lint` to check for pylint errors. Fix any errors.
5. Run `tox` to run the entire test suite including coverage. Fix any errors.
6. If your code touches communication with an ACME server/Boulder, you should run the integration tests, see *integration*.
7. Submit the PR.

3.6 Updating the documentation

In order to generate the Sphinx documentation, run the following commands:

```
make -C docs clean html
```

This should generate documentation in the `docs/_build/html` directory.

3.7 Other methods for running the client

3.7.1 Vagrant

If you are a Vagrant user, Let's Encrypt comes with a Vagrantfile that automates setting up a development environment in an Ubuntu 14.04 LTS VM. To set it up, simply run `vagrant up`. The repository is synced to `/vagrant`, so you can get started with:

```
vagrant ssh
cd /vagrant
./venv/bin/pip install -r requirements.txt .[dev,docs,testing]
sudo ./venv/bin/letsencrypt
```

Support for other Linux distributions coming soon.

Note: Unfortunately, Python distutils and, by extension, setup.py and tox, use hard linking quite extensively. Hard linking is not supported by the default sync filesystem in Vagrant. As a result, all actions with these commands are *significantly slower* in Vagrant. One potential fix is to use NFS (related issue).

3.7.2 Docker

OSX users will probably find it easiest to set up a Docker container for development. Let's Encrypt comes with a Dockerfile (`Dockerfile-dev`) for doing so. To use Docker on OSX, install and setup docker-machine using the instructions at https://docs.docker.com/installation/mac/.

To build the development Docker image:

```
docker build -t letsencrypt -f Dockerfile-dev .
```

Now run tests inside the Docker image:

```
docker run -it letsencrypt bash
cd src
tox -e py27
```

3.8 Notes on OS dependencies

OS level dependencies are managed by scripts in `bootstrap`. Some notes are provided here mainly for the *developers* reference.

In general:

- `sudo` is required as a suggested way of running privileged process
- Augeas is required for the Python bindings
- `virtualenv` and `pip` are used for managing other python library dependencies

3.8.1 Ubuntu

```
sudo ./bootstrap/ubuntu.sh
```

3.8.2 Debian

```
sudo ./bootstrap/debian.sh
```

For squeeze you will need to:

- Use `virtualenv --no-site-packages -p python` instead of `-p python2`.

3.8.3 Mac OSX

```
./bootstrap/mac.sh
```

3.8.4 Fedora

```
sudo ./bootstrap/fedora.sh
```

3.8.5 Centos 7

```
sudo ./bootstrap/centos.sh
```

3.8.6 FreeBSD

```
sudo ./bootstrap/freebsd.sh
```

Bootstrap script for FreeBSD uses `pkg` for package installation, i.e. it does not use ports.

FreeBSD by default uses `tcsh`. In order to activate virtualenv (see below), you will need a compatible shell, e.g. `pkg install bash && bash`.

CHAPTER 4

Packaging Guide

Documentation can be found at https://github.com/letsencrypt/letsencrypt/wiki/Packaging.

CHAPTER 5

API Documentation

5.1 letsencrypt.account

Creates ACME accounts for server.

class letsencrypt.account.**Account**(*regr*, *key*, *meta=None*)
Bases: object

ACME protocol registration.

Variables

- **regr** (*RegistrationResource*) – Registration Resource
- **key** (*JWK*) – Authorized Account Key
- ***Meta*** – Account metadata
- **id** (*str*) – Globally unique account identifier.

class Meta(***kwargs*)
Bases: acme.jose.json_util.JSONObjectWithFields

Account metadata

Variables

- **creation_dt** (*datetime.datetime*) – Creation date and time (UTC).
- **creation_host** (*str*) – FQDN of host, where account has been created.

Note: creation_dt and creation_host are useful in cross-machine migration scenarios.

Account.**slug**
Short account identification string, useful for UI.

letsencrypt.account.**report_new_account**(*acc*, *config*)
Informs the user about their new Let's Encrypt account.

class letsencrypt.account.**AccountMemoryStorage**(*initial_accounts=None*)
Bases: *letsencrypt.interfaces.AccountStorage*

In-memory account strage.

class letsencrypt.account.**AccountFileStorage**(*config*)
Bases: *letsencrypt.interfaces.AccountStorage*

Accounts file storage.

Variables **config** (*IConfig*) – Client configuration

5.2 letsencrypt.achallenges

Client annotated ACME challenges.

Please use names such as `achall` to distiguish from variables "of type" `acme.challenges.Challenge` (denoted by `chall`) and `ChallengeBody` (denoted by `challb`):

```
from acme import challenges
from acme import messages
from letsencrypt import achallenges

chall = challenges.DNS(token='foo')
challb = messages.ChallengeBody(chall=chall)
achall = achallenges.DNS(chall=challb, domain='example.com')
```

Note, that all annotated challenges act as a proxy objects:

```
achall.token == challb.token
```

class letsencrypt.achallenges.**AnnotatedChallenge**(***kwargs*)
Bases: acme.jose.util.ImmutableMap

Client annotated challenge.

Wraps around server provided challenge and annotates with data useful for the client.

Variables **challb** – Wrapped `ChallengeBody`.

class letsencrypt.achallenges.**DVSNI**(***kwargs*)
Bases: *letsencrypt.achallenges.AnnotatedChallenge*

Client annotated "dvsni" ACME challenge.

Variables **account_key** (*JWK*) – Authorized Account Key

acme_type
alias of *DVSNI*

gen_cert_and_response(*key=None*, *bits=2048*, *alg=RS256*)
Generate a DVSNI cert and response.

Parameters

- **key** (*OpenSSL.crypto.PKey*) – Private key used for certificate generation. If none provided, a fresh key will be generated.
- **bits** (*int*) – Number of bits for fresh key generation.
- **alg** (*JWAAlgorithm*) –

Returns `(response, cert_pem, key_pem)` tuple, where `response` is an instance of `acme.challenges.DVSNIResponse`, `cert` is a certificate (`OpenSSL.crypto.X509`) and `key` is a private key (`OpenSSL.crypto.PKey`).

Return type tuple

class letsencrypt.achallenges.**SimpleHTTP**(***kwargs*)
Bases: *letsencrypt.achallenges.AnnotatedChallenge*

Client annotated "simpleHttp" ACME challenge.

acme_type
alias of *SimpleHTTP*

gen_response_and_validation(*tls*)
Generates a SimpleHTTP response and validation.

Parameters **tls** (*bool*) – True if TLS should be used

Returns (response, validation) tuple, where response is an instance of acme.challenges.SimpleHTTPResponse and validation is an instance of acme.challenges.SimpleHTTPProvisionedResource.

Return type tuple

class letsencrypt.achallenges.**DNS**(**kwargs*)
Bases: *letsencrypt.achallenges.AnnotatedChallenge*

Client annotated "dns" ACME challenge.

acme_type
alias of *DNS*

class letsencrypt.achallenges.**RecoveryContact**(**kwargs*)
Bases: *letsencrypt.achallenges.AnnotatedChallenge*

Client annotated "recoveryContact" ACME challenge.

acme_type
alias of *RecoveryContact*

class letsencrypt.achallenges.**ProofOfPossession**(**kwargs*)
Bases: *letsencrypt.achallenges.AnnotatedChallenge*

Client annotated "proofOfPossession" ACME challenge.

acme_type
alias of *ProofOfPossession*

5.3 letsencrypt.auth_handler

ACME AuthHandler.

class letsencrypt.auth_handler.**AuthHandler**(*dv_auth*, *cont_auth*, *acme*, *account*)
Bases: object

ACME Authorization Handler for a client.

Variables

- **dv_auth** – Authenticator capable of solving DVChallenge types
- **cont_auth** – Authenticator capable of solving ContinuityChallenge types
- **acme** (*acme.client.Client*) – ACME client API.
- ***account*** – Client's Account
- **authzr** (*dict*) – ACME Authorization Resource dict where keys are domains and values are acme.messages.AuthorizationResource
- **dv_c** (*list*) – DV challenges in the form of *letsencrypt.achallenges.AnnotatedChallenge*
- **cont_c** (*list*) – Continuity challenges in the form of *letsencrypt.achallenges.AnnotatedChallenge*

get_authorizations(*domains*, *best_effort=False*)
Retrieve all authorizations for challenges.

Parameters

- **domains** (*set*) – Domains for authorization
- **best_effort** (*bool*) – Whether or not all authorizations are required (this is useful in renewal)

Returns tuple of lists of authorization resources. Takes the form of (`completed`, `failed`)

Return type tuple

Raises .AuthorizationError If unable to retrieve all authorizations

_choose_challenges(*domains*)
Retrieve necessary challenges to satisfy server.

_solve_challenges()
Get Responses for challenges from authenticators.

_respond(*cont_resp*, *dv_resp*, *best_effort*)
Send/Receive confirmation of all challenges.

Note: This method also cleans up the auth_handler state.

_send_responses(*achalls*, *resps*, *chall_update*)
Send responses and make sure errors are handled.

Parameters **chall_update** (*dict*) – parameter that is updated to hold authzr -> list of outstanding solved annotated challenges

_poll_challenges(*chall_update*, *best_effort*, *min_sleep=3*, *max_rounds=15*)
Wait for all challenge results to be determined.

_handle_check(*domain*, *achalls*)
Returns tuple of ('completed', 'failed').

_find_updated_challb(*authzr*, *achall*)
Find updated challenge body within Authorization Resource.

Warning: This assumes only one instance of type of challenge in each challenge resource.

Parameters

- **authzr** (*AuthorizationResource*) – Authorization Resource
- **achall** (AnnotatedChallenge) – Annotated challenge for which to get status

_get_chall_pref(*domain*)
Return list of challenge preferences.

Parameters **domain** (*str*) – domain for which you are requesting preferences

_cleanup_challenges(*achall_list=None*)
Cleanup challenges.

If achall_list is not provided, cleanup all achallenges.

verify_authzr_complete()
Verifies that all authorizations have been decided.

Returns Whether all authzr are complete

Return type bool

_challenge_factory(*domain*, *path*)
Construct Namedtuple Challenges

Parameters

- **domain** (*str*) – domain of the enrollee
- **path** (*list*) – List of indices from `challenges`.

Returns dv_chall, list of DVChallenge type `letsencrypt.achallenges.Indexed` cont_chall, list of ContinuityChallenge type `letsencrypt.achallenges.Indexed`

Return type tuple

Raises .errors.Error if challenge type is not recognized

letsencrypt.auth_handler.**challb_to_achall**(*challb*, *account_key*, *domain*)
Converts a ChallengeBody object to an AnnotatedChallenge.

Parameters

- **challb** (*ChallengeBody*) – ChallengeBody
- **account_key** (*JWK*) – Authorized Account Key
- **domain** (*str*) – Domain of the challb

Returns Appropriate AnnotatedChallenge

Return type *letsencrypt.achallenges.AnnotatedChallenge*

letsencrypt.auth_handler.**gen_challenge_path**(*challbs*, *preferences*, *combinations*)
Generate a plan to get authority over the identity.

Todo

This can be possibly be rewritten to use resolved_combinations.

Parameters

- **challbs** (*tuple*) – A tuple of challenges (`acme.messages.Challenge`) from `acme.messages.AuthorizationResource` to be fulfilled by the client in order to prove possession of the identifier.
- **preferences** (*list*) – List of challenge preferences for domain (`acme.challenges.Challenge` subclasses)
- **combinations** (*tuple*) – A collection of sets of challenges from `acme.messages.Challenge`, each of which would be sufficient to prove possession of the identifier.

Returns tuple of indices from `challenges`.

Return type tuple

Raises *letsencrypt.errors.AuthorizationError* If a path cannot be created that satisfies the CA given the preferences and combinations.

letsencrypt.auth_handler.**_find_smart_path**(*challbs*, *preferences*, *combinations*)
Find challenge path with server hints.

Can be called if combinations is included. Function uses a simple ranking system to choose the combo with the lowest cost.

letsencrypt.auth_handler.**_find_dumb_path**(*challbs*, *preferences*)
Find challenge path without server hints.

Should be called if the combinations hint is not included by the server. This function returns the best path that does not contain multiple mutually exclusive challenges.

letsencrypt.auth_handler.**mutually_exclusive**(*obj1*, *obj2*, *groups*, *different=False*)
Are two objects mutually exclusive?

letsencrypt.auth_handler.**is_preferred**(*offered_challb*, *satisfied*, *exclusive_groups=frozenset([frozenset([<class 'acme.challenges.DVSNI'>, <class 'acme.challenges.SimpleHTTP'>])])*)
Return whether or not the challenge is preferred in path.

letsencrypt.auth_handler.**_report_failed_challs**(*failed_achalls*)
Notifies the user about failed challenges.

Parameters **failed_achalls** (*set*) – A set of failed *letsencrypt.achallenges.AnnotatedChallenge*.

letsencrypt.auth_handler.**_generate_failed_chall_msg**(*failed_achalls*)
Creates a user friendly error message about failed challenges.

Parameters **failed_achalls** (*list*) – A list of failed *letsencrypt.achallenges.AnnotatedChallenge* with the same error type.

Returns A formatted error message for the client.

Return type str

5.4 letsencrypt.client

Let's Encrypt client API.

letsencrypt.client.**register**(*config*, *account_storage*, *tos_cb=None*)
Register new account with an ACME CA.

This function takes care of generating fresh private key, registering the account, optionally accepting CA Terms of Service and finally saving the account. It should be called prior to initialization of *Client*, unless account has already been created.

Parameters

- **config** (*IConfig*) – Client configuration.
- **account_storage** (AccountStorage) – Account storage where newly registered account will be saved to. Save happens only after TOS acceptance step, so any account private keys or `RegistrationResource` will not be persisted if `tos_cb` returns `False`.
- **tos_cb** – If ACME CA requires the user to accept a Terms of Service before registering account, client action is necessary. For example, a CLI tool would prompt the user acceptance. `tos_cb` must be a callable that should accept `RegistrationResource` and return a `bool`: `True` iff the Terms of Service present in the contained `Registration.terms_of_service` is accepted by the client, and `False` otherwise. `tos_cb` will be called only if the client acction is necessary, i.e. when `terms_of_service is not None`. This argument is optional, if not supplied it will default to automatic acceptance!

Raises

- ***letsencrypt.errors.Error*** – In case of any client problems, in particular registration failure, or unaccepted Terms of Service.
- **acme.errors.Error** – In case of any protocol problems.

Returns Newly registered and saved account, as well as protocol API handle (should be used in *Client* initialization).

Return type `tuple` of *Account* and `acme.client.Client`

class `letsencrypt.client.`**Client**(*config*, *account_*, *dv_auth*, *installer*, *acme=None*)
Bases: `object`

ACME protocol client.

Variables

- **config** (*IConfig*) – Client configuration.
- ***account*** (Account) – Account registered with *register*.
- ***auth_handler*** (AuthHandler) – Authorizations handler that will dispatch DV and Continuity challenges to appropriate authenticators (providing `IAuthenticator` interface).
- **dv_auth** (*IAuthenticator*) – Prepared (`IAuthenticator.prepare`) authenticator that can solve the `constants.DV_CHALLENGES`.
- **installer** (*IInstaller*) – Installer.
- **acme** (*acme.client.Client*) – Optional ACME client API handle. You might already have one from *register*.

_obtain_certificate(*domains*, *csr*)
Obtain certificate.

Internal function with precondition that `domains` are consistent with identifiers present in the `csr`.

Parameters

- **domains** (*list*) – Domain names.
- **csr** (le_util.CSR) – DER-encoded Certificate Signing Request. The key used to generate this CSR can be different than `authkey`.

Returns `CertificateResource` and certificate chain (as returned by `fetch_chain`).

Return type tuple

obtain_certificate_from_csr(*csr*)
Obtain certficiate from CSR.

Parameters **csr** (le_util.CSR) – DER-encoded Certificate Signing Request.

Returns `CertificateResource` and certificate chain (as returned by `fetch_chain`).

Return type tuple

obtain_certificate(*domains*)
Obtains a certificate from the ACME server.

register must be called before *obtain_certificate*

Parameters **domains** (*set*) – domains to get a certificate

Returns `CertificateResource`, certificate chain (as returned by `fetch_chain`), and newly generated private key (*le_util.Key*) and DER-encoded Certificate Signing Request (*le_util.CSR*).

Return type tuple

obtain_and_enroll_certificate(*domains*, *plugins*)
Obtain and enroll certificate.

Get a new certificate for the specified domains using the specified authenticator and installer, and then create a new renewable lineage containing it.

Parameters

- **domains** (*list*) – Domains to request.
- **plugins** – A PluginsFactory object.

Returns A new `letsencrypt.storage.RenewableCert` instance referred to the enrolled cert lineage, or False if the cert could not be obtained.

save_certificate(*certr*, *chain_cert*, *cert_path*, *chain_path*, *fullchain_path*)
Saves the certificate received from the ACME server.

Parameters

- **certr** (`acme.messages.Certificate`) – ACME "certificate" resource.
- **chain_cert** (*list*) –
- **cert_path** (*str*) – Candidate path to a certificate.
- **chain_path** (*str*) – Candidate path to a certificate chain.
- **fullchain_path** (*str*) – Candidate path to a full cert chain.

Returns cert_path, chain_path, and fullchain_path as absolute paths to the actual files

Return type `tuple` of `str`

Raises IOError If unable to find room to write the cert files

deploy_certificate(*domains*, *privkey_path*, *cert_path*, *chain_path*, *fullchain_path*)
Install certificate

Parameters

- **domains** (*list*) – list of domains to install the certificate
- **privkey_path** (*str*) – path to certificate private key
- **cert_path** (*str*) – certificate file path (optional)
- **chain_path** (*str*) – chain file path

enhance_config(*domains*, *redirect=None*)
Enhance the configuration.

Todo

This needs to handle the specific enhancements offered by the installer. We will also have to find a method to pass in the chosen values efficiently.

Parameters

- **domains** (*list*) – list of domains to configure
- **redirect** (*bool or None*) – If traffic should be forwarded from HTTP to HTTPS.

Raises .errors.Error if no installer is specified in the client.

redirect_to_ssl (*domains*)
Redirect all traffic from HTTP to HTTPS

Parameters **vhost** (*letsencrypt.interfaces.IInstaller*) – list of ssl_vhosts

letsencrypt.client.**validate_key_csr** (*privkey*, *csr=None*)
Validate Key and CSR files.

Verifies that the client key and csr arguments are valid and correspond to one another. This does not currently check the names in the CSR due to the inability to read SANs from CSRs in python crypto libraries.

If csr is left as None, only the key will be validated.

Parameters

- **privkey** (*letsencrypt.le_util.Key*) – Key associated with CSR
- **csr** (le_util.CSR) – CSR

Raises .errors.Error when validation fails

letsencrypt.client.**rollback** (*default_installer*, *checkpoints*, *config*, *plugins*)
Revert configuration the specified number of checkpoints.

Parameters

- **checkpoints** (*int*) – Number of checkpoints to revert.
- **config** (*letsencrypt.interfaces.IConfig*) – Configuration.

letsencrypt.client.**view_config_changes** (*config*)
View checkpoints and associated configuration changes.

Note: This assumes that the installation is using a Reverter object.

Parameters **config** (*letsencrypt.interfaces.IConfig*) – Configuration.

letsencrypt.client.**_save_chain** (*chain_pem*, *chain_path*)
Saves chain_pem at a unique path based on chain_path.

Parameters

- **chain_pem** (*str*) – certificate chain in PEM format
- **chain_path** (*str*) – candidate path for the cert chain

Returns absolute path to saved cert chain

Return type str

5.5 letsencrypt.configuration

Let's Encrypt user-supplied configuration.

class letsencrypt.configuration.**NamespaceConfig** (*namespace*)
Bases: object

Configuration wrapper around argparse.Namespace.

For more documentation, including available attributes, please see *letsencrypt.interfaces.IConfig*. However, note that the following attributes are dynamically resolved using *work_dir* and relative paths defined in *letsencrypt.constants*:

- accounts_dir
- csr_dir
- in_progress_dir
- key_dir
- renewer_config_file
- temp_checkpoint_dir

Variables **namespace** – Namespace typically produced by argparse.ArgumentParser.parse_args().

server_path
File path based on server.

class letsencrypt.configuration.**RenewerConfiguration**(*namespace*)
Bases: object

Configuration wrapper for renewer.

5.6 letsencrypt.constants

Let's Encrypt constants.

letsencrypt.constants.**SETUPTOOLS_PLUGINS_ENTRY_POINT = 'letsencrypt.plugins'**
Setuptools entry point group name for plugins.

letsencrypt.constants.**CLI_DEFAULTS = {'strict_permissions': False, 'verbose_count': -3, 'no_verify_ssl': False, 'dvsn**
Defaults for CLI flags and IConfig attributes.

letsencrypt.constants.**RENEWER_DEFAULTS = {'renew_before_expiry': '30 days', 'deploy_before_expiry': '20 days', '**
Defaults for renewer script.

letsencrypt.constants.**EXCLUSIVE_CHALLENGES = frozenset([frozenset([<class 'acme.challenges.DVSNI'>, <class 'ac**
Mutually exclusive challenges.

letsencrypt.constants.**ENHANCEMENTS = ['redirect', 'http-header', 'ocsp-stapling', 'spdy']**
List of possible *letsencrypt.interfaces.IInstaller* enhancements.

List of expected options parameters: - redirect: None - http-header: TODO - ocsp-stapling: TODO - spdy: TODO

letsencrypt.constants.**ARCHIVE_DIR = 'archive'**
Archive directory, relative to IConfig.config_dir.

letsencrypt.constants.**CONFIG_DIRS_MODE = 493**
Directory mode for .IConfig.config_dir et al.

letsencrypt.constants.**ACCOUNTS_DIR = 'accounts'**
Directory where all accounts are saved.

letsencrypt.constants.**BACKUP_DIR = 'backups'**
Directory (relative to IConfig.work_dir) where backups are kept.

letsencrypt.constants.**CSR_DIR = 'csr'**
See *IConfig.csr_dir*.

letsencrypt.constants.**IN_PROGRESS_DIR = 'IN_PROGRESS'**
Directory used before a permanent checkpoint is finalized (relative to IConfig.work_dir).

letsencrypt.constants.**KEY_DIR = 'keys'**
Directory (relative to IConfig.config_dir) where keys are saved.

letsencrypt.constants.**LIVE_DIR = 'live'**
Live directory, relative to IConfig.config_dir.

letsencrypt.constants.**TEMP_CHECKPOINT_DIR = 'temp_checkpoint'**
Temporary checkpoint directory (relative to IConfig.work_dir).

letsencrypt.constants.**RENEWAL_CONFIGS_DIR = 'renewal'**
Renewal configs directory, relative to IConfig.config_dir.

letsencrypt.constants.**RENEWER_CONFIG_FILENAME = 'renewer.conf'**
Renewer config file name (relative to IConfig.config_dir).

5.7 letsencrypt.continuity_auth

Continuity Authenticator

class letsencrypt.continuity_auth.**ContinuityAuthenticator**(*config*, *installer*)
Bases: object

IAuthenticator for ContinuityChallenge class challenges.

Variables **proof_of_pos** – Performs "proofOfPossession" challenges.

get_chall_pref(*unused_domain*)
Return list of challenge preferences.

perform(*achalls*)
Perform client specific challenges for IAuthenticator

cleanup(*achalls*)
Cleanup call for IAuthenticator.

5.8 letsencrypt.crypto_util

Let's Encrypt client crypto utility functions.

Todo

Make the transition to use PSS rather than PKCS1_v1_5 when the server is capable of handling the signatures.

letsencrypt.crypto_util.**init_save_key**(*key_size*, *key_dir*, *keyname='key-letsencrypt.pem'*)
Initializes and saves a privkey.

Inits key and saves it in PEM format on the filesystem.

Note: keyname is the attempted filename, it may be different if a file already exists at the path.

Parameters

- **key_size** (*int*) – RSA key size in bits
- **key_dir** (*str*) – Key save directory.
- **keyname** (*str*) – Filename of key

Returns Key

Return type *letsencrypt.le_util.Key*

Raises ValueError If unable to generate the key given key_size.

letsencrypt.crypto_util.**init_save_csr**(*privkey*, *names*, *path*, *csrname='csr-letsencrypt.pem'*)

Initialize a CSR with the given private key.

Parameters

- **privkey** (*letsencrypt.le_util.Key*) – Key to include in the CSR
- **names** (*set*) – `str` names to include in the CSR
- **path** (*str*) – Certificate save directory.

Returns CSR

Return type *letsencrypt.le_util.CSR*

letsencrypt.crypto_util.**make_csr**(*key_str*, *domains*)

Generate a CSR.

Parameters

- **key_str** (*str*) – PEM-encoded RSA key.
- **domains** (*list*) – Domains included in the certificate.

Todo

Detect duplicates in `domains`? Using a set doesn't preserve order...

Returns new CSR in PEM and DER form containing all domains

Return type tuple

letsencrypt.crypto_util.**valid_csr**(*csr*)

Validate CSR.

Check if `csr` is a valid CSR for the given domains.

Parameters **csr** (*str*) – CSR in PEM.

Returns Validity of CSR.

Return type bool

letsencrypt.crypto_util.**csr_matches_pubkey**(*csr*, *privkey*)

Does private key correspond to the subject public key in the CSR?

Parameters

- **csr** (*str*) – CSR in PEM.
- **privkey** (*str*) – Private key file contents (PEM)

Returns Correspondence of private key to CSR subject public key.

Return type bool

letsencrypt.crypto_util.**make_key**(*bits*)

Generate PEM encoded RSA key.

Parameters **bits** (*int*) – Number of bits, at least 1024.

Returns new RSA key in PEM form with specified number of bits

Return type str

letsencrypt.crypto_util.**valid_privkey**(*privkey*)
Is valid RSA private key?

Parameters **privkey** (*str*) – Private key file contents in PEM

Returns Validity of private key.

Return type bool

letsencrypt.crypto_util.**pyopenssl_load_certificate**(*data*)
Load PEM/DER certificate.

Raises errors.Error

letsencrypt.crypto_util.**get_sans_from_cert**(*cert*, *typ=1*)
Get a list of Subject Alternative Names from a certificate.

Parameters

- **cert** (*str*) – Certificate (encoded).
- **typ** – `OpenSSL.crypto.FILETYPE_PEM` or `OpenSSL.crypto.FILETYPE_ASN1`

Returns A list of Subject Alternative Names.

Return type list

letsencrypt.crypto_util.**get_sans_from_csr**(*csr*, *typ=1*)
Get a list of Subject Alternative Names from a CSR.

Parameters

- **csr** (*str*) – CSR (encoded).
- **typ** – `OpenSSL.crypto.FILETYPE_PEM` or `OpenSSL.crypto.FILETYPE_ASN1`

Returns A list of Subject Alternative Names.

Return type list

letsencrypt.crypto_util.**dump_pyopenssl_chain**(*chain*, *filetype=1*)
Dump certificate chain into a bundle.

Parameters **chain** (*list*) – List of `OpenSSL.crypto.X509` (or wrapped in `acme.jose.ComparableX509`).

letsencrypt.crypto_util.**notBefore**(*cert_path*)
When does the cert at cert_path start being valid?

Parameters **cert_path** (*str*) – path to a cert in PEM format

Returns the notBefore value from the cert at cert_path

Return type `datetime.datetime`

letsencrypt.crypto_util.**notAfter**(*cert_path*)
When does the cert at cert_path stop being valid?

Parameters **cert_path** (*str*) – path to a cert in PEM format

Returns the notAfter value from the cert at cert_path

Return type `datetime.datetime`

letsencrypt.crypto_util.**_notAfterBefore**(*cert_path*, *method*)
Internal helper function for finding notbefore/notafter.

Parameters

- **cert_path** (*str*) – path to a cert in PEM format
- **method** (*function*) – one of `OpenSSL.crypto.X509.get_notBefore` or `OpenSSL.crypto.X509.get_notAfter`

Returns the notBefore or notAfter value from the cert at cert_path

Return type `datetime.datetime`

5.9 letsencrypt.display

Let's Encrypt display utilities.

5.9.1 letsencrypt.display.util

Let's Encrypt display.

letsencrypt.display.util.**OK = 'ok'**
Display exit code indicating user acceptance.

letsencrypt.display.util.**CANCEL = 'cancel'**
Display exit code for a user canceling the display.

letsencrypt.display.util.**HELP = 'help'**
Display exit code when for when the user requests more help.

class letsencrypt.display.util.**NcursesDisplay**(*width=72*, *height=20*)
Bases: `object`

Ncurses-based display.

notification(*message*, *height=10*, *pause=False*)
Display a notification to the user and wait for user acceptance.

Todo

It probably makes sense to use one of the transient message types for pause. It isn't straightforward how best to approach the matter though given the context of our messages. http://pythondialog.sourceforge.net/doc/widgets.html#displaying-transient-messages

---

Parameters

- **message** (*str*) – Message to display
- **height** (*int*) – Height of the dialog box
- **pause** (*bool*) – Not applicable to NcursesDisplay

menu(*message*, *choices*, *ok_label='OK'*, *cancel_label='Cancel'*, *help_label=''*)
Display a menu.

Parameters

- **message** (*str*) – title of menu

- **choices** (list of tuples (`tag`, `item`) tags must be unique or list of items (tags will be enumerated)) – menu lines, len must be > 0
- **ok_label** (*str*) – label of the OK button
- **help_label** (*str*) – label of the help button

Returns tuple of the form (`code`, `tag`) where `code` - `str` display_util exit code `tag` - `int` index corresponding to the item chosen

Return type tuple

input(*message*)
Display an input box to the user.

Parameters **message** (*str*) – Message to display that asks for input.

Returns tuple of the form (code, string) where `code` - int display exit code `string` - input entered by the user

yesno(*message*, *yes_label='Yes'*, *no_label='No'*)
Display a Yes/No dialog box.

Yes and No label must begin with different letters.

Parameters

- **message** (*str*) – message to display to user
- **yes_label** (*str*) – label on the "yes" button
- **no_label** (*str*) – label on the "no" button

Returns if yes_label was selected

Return type bool

checklist(*message*, *tags*, *default_status=True*)
Displays a checklist.

Parameters

- **message** – Message to display before choices
- **tags** (*list*) – where each is of type `str` len(tags) > 0
- **default_status** (*bool*) – If True, items are in a selected state by default.

Returns tuple of the form (code, list_tags) where `code` - int display exit code `list_tags` - list of str tags selected by the user

class `letsencrypt.display.util.`**FileDisplay**(*outfile*)
Bases: `object`

File-based display.

notification(*message*, *height=10*, *pause=True*)
Displays a notification and waits for user acceptance.

Parameters

- **message** (*str*) – Message to display
- **height** (*int*) – No effect for FileDisplay
- **pause** (*bool*) – Whether or not the program should pause for the user's confirmation

menu(*message*, *choices*, *ok_label=''*, *cancel_label=''*, *help_label=''*)
Display a menu.

Todo

This doesn't enable the help label/button (I wasn't sold on any interface I came up with for this). It would be a nice feature

Parameters

- **message** (*str*) – title of menu
- **choices** (*list of tuples (tag, item) or list of descriptions (tags will be enumerated)*) – Menu lines, len must be > 0

Returns tuple of the form (code, tag) where code - int display exit code tag - str corresponding to the item chosen

Return type tuple

input(*message*)
Accept input from the user.

Parameters **message** (*str*) – message to display to the user

Returns tuple of (`code`, *`input`*) where `code` - str display exit code *`input`* - str of the user's input

Return type tuple

yesno(*message*, *yes_label='Yes'*, *no_label='No'*)
Query the user with a yes/no question.

Yes and No label must begin with different letters, and must contain at least one letter each.

Parameters

- **message** (*str*) – question for the user
- **yes_label** (*str*) – Label of the "Yes" parameter
- **no_label** (*str*) – Label of the "No" parameter

Returns True for "Yes", False for "No"

Return type bool

checklist(*message*, *tags*, *default_status=True*)
Display a checklist.

Parameters

- **message** (*str*) – Message to display to user
- **tags** (*list*) – `str` tags to select, len(tags) > 0
- **default_status** (*bool*) – Not used for FileDisplay

Returns tuple of (`code`, `tags`) where `code` - str display exit code `tags` - list of selected tags

Return type tuple

_scrub_checklist_input(*indices*, *tags*)
Validate input and transform indices to appropriate tags.

Parameters

- **indices** (*list*) – input
- **tags** (*list*) – Original tags of the checklist

Returns valid tags the user selected

Return type `list` of `str`

_print_menu(*message*, *choices*)
Print a menu on the screen.

Parameters

- **message** (*str*) – title of menu
- **choices** (*list of tuples (tag, item) or list of descriptions (tags will be enumerated)*) – Menu lines

_wrap_lines(*msg*)
Format lines nicely to 80 chars.

Parameters **msg** (*str*) – Original message

Returns Formatted message respecting newlines in message

Return type str

_get_valid_int_ans(*max_*)
Get a numerical selection.

Parameters **max** (*int*) – The maximum entry (len of choices), must be positive

Returns tuple of the form (`code`, `selection`) where `code` - str display exit code ('ok' or cancel') `selection` - int user's selection

Return type tuple

`letsencrypt.display.util.`**separate_list_input**(*input_*)
Separate a comma or space separated list.

Parameters **input** (*str*) – input from the user

Returns strings

Return type list

`letsencrypt.display.util.`**_parens_around_char**(*label*)
Place parens around first character of label.

Parameters **label** (*str*) – Must contain at least one character

5.9.2 letsencrypt.display.ops

Contains UI methods for LE user operations.

`letsencrypt.display.ops.`**choose_plugin**(*prepared*, *question*)
Allow the user to choose their plugin.

Parameters

- **prepared** (*list*) – List of *PluginEntryPoint*.
- **question** (*str*) – Question to be presented to the user.

Returns Plugin entry point chosen by the user.

Return type *PluginEntryPoint*

letsencrypt.display.ops.**pick_plugin**(*config*, *default*, *plugins*, *question*, *ifaces*)
Pick plugin.

Parameters

- **letsencrypt.interfaces.IConfig** – Configuration
- **default** (*str*) – Plugin name supplied by user or None.
- **plugins** (letsencrypt.plugins.disco.PluginsRegistry) – All plugins registered as entry points.
- **question** (*str*) – Question to be presented to the user in case multiple candidates are found.
- **ifaces** (*list*) – Interfaces that plugins must provide.

Returns Initialized plugin.

Return type IPlugin

letsencrypt.display.ops.**pick_authenticator**(*config*, *default*, *plugins*, *question="How would you like to authenticate with the Let's Encrypt CA?"*)
Pick authentication plugin.

letsencrypt.display.ops.**pick_installer**(*config*, *default*, *plugins*, *question='How would you like to install certificates?'*)
Pick installer plugin.

letsencrypt.display.ops.**pick_configurator**(*config*, *default*, *plugins*, *question='How would you like to authenticate and install certificates?'*)
Pick configurator plugin.

letsencrypt.display.ops.**get_email**()
Prompt for valid email address.

Returns Email or None if cancelled by user.

Return type str

letsencrypt.display.ops.**choose_account**(*accounts*)
Choose an account.

Parameters **accounts** (*list*) – Containing at least one *Account*

letsencrypt.display.ops.**choose_names**(*installer*)
Display screen to select domains to validate.

Parameters **installer** (*letsencrypt.interfaces.IInstaller*) – An installer object

Returns List of selected names

Return type list of str

letsencrypt.display.ops.**_filter_names**(*names*)
Determine which names the user would like to select from a list.

Parameters **names** (*list*) – domain names

Returns tuple of the form (code, names) where code - str display exit code names - list of names selected

Return type tuple

letsencrypt.display.ops.**_choose_names_manually**()
Manually input names for those without an installer.

letsencrypt.display.ops.**success_installation**(*domains*)
Display a box confirming the installation of HTTPS.

Todo

This should be centered on the screen

Parameters **domains** (*list*) – domain names which were enabled

letsencrypt.display.ops.**success_renewal**(*domains*)
Display a box confirming the renewal of an existing certificate.

Todo

This should be centered on the screen

Parameters **domains** (*list*) – domain names which were renewed

letsencrypt.display.ops.**_gen_ssl_lab_urls**(*domains*)
Returns a list of urls.

Parameters **domains** (*list*) – Each domain is a 'str'

letsencrypt.display.ops.**_gen_https_names**(*domains*)
Returns a string of the https domains.

Domains are formatted nicely with https:// prepended to each.

Parameters **domains** (*list*) – Each domain is a 'str'

5.9.3 letsencrypt.display.enhancements

Let's Encrypt Enhancement Display

letsencrypt.display.enhancements.**ask**(*enhancement*)
Display the enhancement to the user.

Parameters **enhancement** (*str*) – One of the letsencrypt.CONFIG.ENHANCEMENTS enhancements

Returns True if feature is desired, False otherwise

Return type bool

Raises .errors.Error if the enhancement provided is not supported

letsencrypt.display.enhancements.**redirect_by_default**()
Determines whether the user would like to redirect to HTTPS.

Returns True if redirect is desired, False otherwise

Return type bool

5.10 letsencrypt.errors

Let's Encrypt client errors.

exception letsencrypt.errors.**Error**
Bases: exceptions.Exception

Generic Let's Encrypt client error.

exception letsencrypt.errors.**AccountStorageError**
Bases: *letsencrypt.errors.Error*

Generic *AccountStorage* error.

exception letsencrypt.errors.**AccountNotFound**
Bases: *letsencrypt.errors.AccountStorageError*

Account not found error.

exception letsencrypt.errors.**ReverterError**
Bases: *letsencrypt.errors.Error*

Let's Encrypt Reverter error.

exception letsencrypt.errors.**SubprocessError**
Bases: *letsencrypt.errors.Error*

Subprocess handling error.

exception letsencrypt.errors.**CertStorageError**
Bases: *letsencrypt.errors.Error*

Generic CertStorage error.

exception letsencrypt.errors.**AuthorizationError**
Bases: *letsencrypt.errors.Error*

Authorization error.

exception letsencrypt.errors.**FailedChallenges**(*failed_achalls*)
Bases: *letsencrypt.errors.AuthorizationError*

Failed challenges error.

Variables **failed_achalls** (*set*) – Failed *AnnotatedChallenge* instances.

exception letsencrypt.errors.**ContAuthError**
Bases: *letsencrypt.errors.AuthorizationError*

Let's Encrypt Continuity Authenticator error.

exception letsencrypt.errors.**DvAuthError**
Bases: *letsencrypt.errors.AuthorizationError*

Let's Encrypt DV Authenticator error.

exception letsencrypt.errors.**DvsniError**
Bases: *letsencrypt.errors.DvAuthError*

Let's Encrypt DVSNI error.

exception letsencrypt.errors.**PluginError**
Bases: *letsencrypt.errors.Error*

Let's Encrypt Plugin error.

exception letsencrypt.errors.**PluginSelectionError**
Bases: *letsencrypt.errors.Error*

A problem with plugin/configurator selection or setup

exception letsencrypt.errors.**NoInstallationError**
Bases: *letsencrypt.errors.PluginError*

Let's Encrypt No Installation error.

exception letsencrypt.errors.**MisconfigurationError**
Bases: *letsencrypt.errors.PluginError*

Let's Encrypt Misconfiguration error.

exception letsencrypt.errors.**NotSupportedError**
Bases: *letsencrypt.errors.PluginError*

Let's Encrypt Plugin function not supported error.

exception letsencrypt.errors.**RevokerError**
Bases: *letsencrypt.errors.Error*

Let's Encrypt Revoker error.

exception letsencrypt.errors.**StandaloneBindError**(*socket_error*, *port*)
Bases: *letsencrypt.errors.Error*

Standalone plugin bind error.

5.11 letsencrypt

Let's Encrypt client.

5.12 letsencrypt.interfaces

Let's Encrypt client interfaces.

class letsencrypt.interfaces.**AccountStorage**
Bases: object

Accounts storage interface.

find_all()
Find all accounts.

Returns All found accounts.

Return type list

load(*account_id*)
Load an account by its id.

Raises

- **.AccountNotFound** – if account could not be found
- **.AccountStorageError** – if account could not be loaded

save(*account*)
Save account.

Raises .AccountStorageError if account could not be saved

interface `letsencrypt.interfaces.`**IPluginFactory**
IPlugin factory.

Objects providing this interface will be called without satisfying any entry point "extras" (extra dependencies) you might have defined for your plugin, e.g (excerpt from `setup.py` script):

```
setup(
    ...
    entry_points={
        'letsencrypt.plugins': [
            'name=example_project.plugin[plugin_deps]',
        ],
    },
    extras_require={
        'plugin_deps': ['dep1', 'dep2'],
    }
)
```

Therefore, make sure such objects are importable and usable without extras. This is necessary, because CLI does the following operations (in order):

- loads an entry point,
- calls *inject_parser_options*,
- requires an entry point,
- creates plugin instance (*__call__*).

description
Short plugin description

__call__(*config*, *name*)
Create new *IPlugin*.

Parameters

- **config** (*IConfig*) – Configuration.
- **name** (*str*) – Unique plugin name.

inject_parser_options(*parser*, *name*)
Inject argument parser options (flags).

1. Be nice and prepend all options and destinations with *option_namespace* and `dest_namespace`.
2. Inject options (flags) only. Positional arguments are not allowed, as this would break the CLI.

Parameters

- **parser** (*ArgumentParser*) – (Almost) top-level CLI parser.
- **name** (*str*) – Unique plugin name.

interface `letsencrypt.interfaces.`**IPlugin**
Let's Encrypt plugin.

prepare()
Prepare the plugin.

Finish up any additional initialization.

Raises

- **.PluginError** – when full initialization cannot be completed.
- **.MisconfigurationError** – when full initialization cannot be completed. Plugin will be displayed on a list of available plugins.
- **.NoInstallationError** – when the necessary programs/files cannot be located. Plugin will NOT be displayed on a list of available plugins.
- **.NotSupportedError** – when the installation is recognized, but the version is not currently supported.

more_info()
Human-readable string to help the user.

Should describe the steps taken and any relevant info to help the user decide which plugin to use.

Rtype str

interface letsencrypt.interfaces.**IAuthenticator**
Extends: *letsencrypt.interfaces.IPlugin*

Generic Let's Encrypt Authenticator.

Class represents all possible tools processes that have the ability to perform challenges and attain a certificate.

get_chall_pref(*domain*)
Return list of challenge preferences.

Parameters **domain** (*str*) – Domain for which challenge preferences are sought.

Returns List of challenge types (subclasses of acme.challenges.Challenge) with the most preferred challenges first. If a type is not specified, it means the Authenticator cannot perform the challenge.

Return type list

perform(*achalls*)
Perform the given challenge.

Parameters **achalls** (*list*) – Non-empty (guaranteed) list of *AnnotatedChallenge* instances, such that it contains types found within *get_chall_pref()* only.

Returns

List of ACME ChallengeResponse instances or if the Challenge cannot be fulfilled then:

None Authenticator can perform challenge, but not at this time.

False Authenticator will never be able to perform (error).

Return type list of acme.challenges.ChallengeResponse

Raises .PluginError If challenges cannot be performed

cleanup(*achalls*)
Revert changes and shutdown after challenges complete.

Parameters **achalls** (*list*) – Non-empty (guaranteed) list of *AnnotatedChallenge* instances, a subset of those previously passed to *perform()*.

Raises PluginError if original configuration cannot be restored

interface letsencrypt.interfaces.**IConfig**
Let's Encrypt user-supplied configuration.

> **Warning:** The values stored in the configuration have not been filtered, stripped or sanitized.

server
ACME Directory Resource URI.

email
Email used for registration and recovery contact.

rsa_key_size
Size of the RSA key.

config_dir
Configuration directory.

work_dir
Working directory.

accounts_dir
Directory where all account information is stored.

backup_dir
Configuration backups directory.

csr_dir
Directory where newly generated Certificate Signing Requests (CSRs) are saved.

in_progress_dir
Directory used before a permanent checkpoint is finalized.

key_dir
Keys storage.

temp_checkpoint_dir
Temporary checkpoint directory.

renewer_config_file
Location of renewal configuration file.

no_verify_ssl
Disable SSL certificate verification.

dvsni_port
Port number to perform DVSNI challenge. Boulder in testing mode defaults to 5001.

simple_http_port
Port used in the SimpleHttp challenge.

interface `letsencrypt.interfaces.`**IInstaller**
Extends: *letsencrypt.interfaces.IPlugin*

Generic Let's Encrypt Installer Interface.

Represents any server that an X509 certificate can be placed.

get_all_names()
Returns all names that may be authenticated.

Return type `list` of `str`

deploy_cert(*domain*, *cert_path*, *key_path*, *chain_path*, *fullchain_path*)
Deploy certificate.

Parameters

- **domain** (*str*) – domain to deploy certificate file
- **cert_path** (*str*) – absolute path to the certificate file
- **key_path** (*str*) – absolute path to the private key file
- **chain_path** (*str*) – absolute path to the certificate chain file
- **fullchain_path** (*str*) – absolute path to the certificate fullchain file (cert plus chain)

Raises .PluginError when cert cannot be deployed

enhance(*domain*, *enhancement*, *options=None*)
Perform a configuration enhancement.

Parameters

- **domain** (*str*) – domain for which to provide enhancement
- **enhancement** (*str*) – An enhancement as defined in *ENHANCEMENTS*
- **options** – Flexible options parameter for enhancement. Check documentation of *ENHANCEMENTS* for expected options for each enhancement.

Raises .PluginError If Enhancement is not supported, or if an error occurs during the enhancement.

supported_enhancements()
Returns a list of supported enhancements.

Returns supported enhancements which should be a subset of *ENHANCEMENTS*

Return type `list` of `str`

get_all_certs_keys()
Retrieve all certs and keys set in configuration.

Returns

tuples with form `[(cert, key, path)]`, where:

- `cert` - str path to certificate file
- `key` - str path to associated key file
- `path` - file path to configuration file

Return type list

save(*title=None*, *temporary=False*)
Saves all changes to the configuration files.

Both title and temporary are needed because a save may be intended to be permanent, but the save is not ready to be a full checkpoint

Parameters

- **title** (*str*) – The title of the save. If a title is given, the configuration will be saved as a new checkpoint and put in a timestamped directory. `title` has no effect if temporary is true.
- **temporary** (*bool*) – Indicates whether the changes made will be quickly reversed in the future (challenges)

Raises .PluginError when save is unsuccessful

rollback_checkpoints(*rollback=1*)
Revert `rollback` number of configuration checkpoints.

Raises .PluginError when configuration cannot be fully reverted

recovery_routine()
Revert configuration to most recent finalized checkpoint.

Remove all changes (temporary and permanent) that have not been finalized. This is useful to protect against crashes and other execution interruptions.

Raises .errors.PluginError If unable to recover the configuration

view_config_changes()
Display all of the LE config changes.

Raises .PluginError when config changes cannot be parsed

config_test()
Make sure the configuration is valid.

Raises .MisconfigurationError when the config is not in a usable state

restart()
Restart or refresh the server content.

Raises .PluginError when server cannot be restarted

interface letsencrypt.interfaces.**IDisplay**
Generic display.

notification(*message*, *height*, *pause*)
Displays a string message

Parameters

- **message** (*str*) – Message to display
- **height** (*int*) – Height of dialog box if applicable
- **pause** (*bool*) – Whether or not the application should pause for confirmation (if available)

menu(*message*, *choices*, *ok_label='OK'*, *cancel_label='Cancel'*, *help_label=''*)
Displays a generic menu.

Parameters

- **message** (*str*) – message to display
- **choices** (list of tuple() or str) – choices
- **ok_label** (*str*) – label for OK button
- **cancel_label** (*str*) – label for Cancel button
- **help_label** (*str*) – label for Help button

Returns tuple of (code, index) where code - str display exit code index - int index of the user's selection

input(*message*)
Accept input from the user.

Parameters **message** (*str*) – message to display to the user

Returns tuple of (code, *input*) where code - str display exit code *input* - str of the user's input

Return type tuple

yesno(*message*, *yes_label='Yes'*, *no_label='No'*)
Query the user with a yes/no question.

Yes and No label must begin with different letters.

Parameters **message** (*str*) – question for the user

Returns True for "Yes", False for "No"

Return type bool

checklist(*message*, *tags*, *default_state*)
Allow for multiple selections from a menu.

Parameters

- **message** (*str*) – message to display to the user
- **tags** (*list*) – where each is of type `str` len(tags) > 0
- **default_status** (*bool*) – If True, items are in a selected state by default.

interface `letsencrypt.interfaces.`**IValidator**
Configuration validator.

certificate(*cert*, *name*, *alt_host=None*, *port=443*)
Verifies the certificate presented at name is cert

Parameters

- **cert** (*OpenSSL.crypto.X509*) – Expected certificate
- **name** (*str*) – Server's domain name
- **alt_host** (*bytes*) – Host to connect to instead of the IP address of host
- **port** (*int*) – Port to connect to

Returns True if the certificate was verified successfully

Return type bool

redirect(*name*, *port=80*, *headers=None*)
Verify redirect to HTTPS

Parameters

- **name** (*str*) – Server's domain name
- **port** (*int*) – Port to connect to
- **headers** (*dict*) – HTTP headers to include in request

Returns True if redirect is successfully enabled

Return type bool

hsts(*name*)
Verify HSTS header is enabled

Parameters **name** (*str*) – Server's domain name

Returns True if HSTS header is successfully enabled

Return type bool

ocsp_stapling(*name*)
Verify ocsp stapling for domain

Parameters **name** (*str*) – Server's domain name

Returns True if ocsp stapling is successfully enabled

Return type bool

interface letsencrypt.interfaces.**IReporter**
Interface to collect and display information to the user.

HIGH_PRIORITY
Used to denote high priority messages

MEDIUM_PRIORITY
Used to denote medium priority messages

LOW_PRIORITY
Used to denote low priority messages

add_message(*self*, *msg*, *priority*, *on_crash=True*)
Adds msg to the list of messages to be printed.

Parameters

- **msg** (*str*) – Message to be displayed to the user.
- **priority** (*int*) – One of HIGH_PRIORITY, MEDIUM_PRIORITY, or LOW_PRIORITY.
- **on_crash** (*bool*) – Whether or not the message should be printed if the program exits abnormally.

print_messages(*self*)
Prints messages to the user and clears the message queue.

5.13 letsencrypt.le_util

Utilities for all Let's Encrypt.

class letsencrypt.le_util.**Key**(*file*, *pem*)
Bases: tuple

_asdict()
Return a new OrderedDict which maps field names to their values

classmethod _make(*iterable*, *new=<built-in method __new__ of type object at 0x9192c0>*, *len=<built-in function len>*)
Make a new Key object from a sequence or iterable

_replace(*_self*, ***kwds*)
Return a new Key object replacing specified fields with new values

file
Alias for field number 0

pem
Alias for field number 1

class letsencrypt.le_util.**CSR**(*file*, *data*, *form*)
Bases: tuple

_asdict()
Return a new OrderedDict which maps field names to their values

classmethod _make(*iterable*, *new=<built-in method __new__ of type object at 0x9192c0>*, *len=<built-in function len>*)
Make a new CSR object from a sequence or iterable

_replace(*_self*, ***kwds*)
Return a new CSR object replacing specified fields with new values

data
Alias for field number 1

file
Alias for field number 0

form
Alias for field number 2

letsencrypt.le_util.**run_script**(*params*)
Run the script with the given params.

Parameters **params** (*list*) – List of parameters to pass to Popen

letsencrypt.le_util.**exe_exists**(*exe*)
Determine whether path/name refers to an executable.

Parameters **exe** (*str*) – Executable path or name

Returns If exe is a valid executable

Return type bool

letsencrypt.le_util.**make_or_verify_dir**(*directory*, *mode=493*, *uid=0*, *strict=False*)
Make sure directory exists with proper permissions.

Parameters

- **directory** (*str*) – Path to a directory.
- **mode** (*int*) – Directory mode.
- **uid** (*int*) – Directory owner.

Raises

- **.errors.Error** – if a directory already exists, but has wrong permissions or owner
- **OSError** – if invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system.

letsencrypt.le_util.**check_permissions**(*filepath*, *mode*, *uid=0*)
Check file or directory permissions.

Parameters

- **filepath** (*str*) – Path to the tested file (or directory).
- **mode** (*int*) – Expected file mode.
- **uid** (*int*) – Expected file owner.

Returns True if `mode` and `uid` match, False otherwise.

Return type bool

letsencrypt.le_util.**safe_open**(*path*, *mode='w'*, *chmod=None*, *buffering=None*)
Safely open a file.

Parameters

- **path** (*str*) – Path to a file.
- **mode** (*str*) – Same os `mode` for `open`.
- **chmod** (*int*) – Same as `mode` for `os.open`, uses Python defaults if `None`.
- **buffering** (*int*) – Same as `bufsize` for `os.fdopen`, uses Python defaults if `None`.

`letsencrypt.le_util.`**unique_file**(*path*, *mode=511*)
Safely finds a unique file.

Parameters

- **path** (*str*) – path/filename.ext
- **mode** (*int*) – File mode

Returns tuple of file object and file name

`letsencrypt.le_util.`**unique_lineage_name**(*path*, *filename*, *mode=511*)
Safely finds a unique file using lineage convention.

Parameters

- **path** (*str*) – directory path
- **filename** (*str*) – proposed filename
- **mode** (*int*) – file mode

Returns tuple of file object and file name (which may be modified from the requested one by appending digits to ensure uniqueness)

Raises OSError if writing files fails for an unanticipated reason, such as a full disk or a lack of permission to write to specified location.

`letsencrypt.le_util.`**safely_remove**(*path*)
Remove a file that may not exist.

`letsencrypt.le_util.`**safe_email**(*email*)
Scrub email address before using it.

5.14 letsencrypt.log

Logging utilities.

class `letsencrypt.log.`**DialogHandler**(*level=0*, *height=20*, *width=68*, *d=None*)
Bases: `logging.Handler`

Logging handler using dialog info box.

Variables

- **height** (*int*) – Height of the info box (without padding).
- **width** (*int*) – Width of the info box (without padding).
- **lines** (*list*) – Lines to be displayed in the info box.
- **d** – Instance of `dialog.Dialog`.

emit(*record*)
Emit message to a dialog info box.

Only show the last (self.height) lines; note that lines can wrap at self.width, so a single line could actually be multiple lines.

5.15 letsencrypt.plugins.common

Plugin common functions.

letsencrypt.plugins.common.**option_namespace**(*name*)
ArgumentParser options namespace (prefix of all options).

letsencrypt.plugins.common.**dest_namespace**(*name*)
ArgumentParser dest namespace (prefix of all destinations).

class letsencrypt.plugins.common.**Plugin**(*config*, *name*)
Bases: object

Generic plugin.

option_namespace
ArgumentParser options namespace (prefix of all options).

option_name(*name*)
Option name (include plugin namespace).

dest_namespace
ArgumentParser dest namespace (prefix of all destinations).

dest(*var*)
Find a destination for given variable `var`.

conf(*var*)
Find a configuration value for variable `var`.

classmethod inject_parser_options(*parser*, *name*)
Inject parser options.

See `inject_parser_options` for docs.

classmethod add_parser_arguments(*add*)
Add plugin arguments to the CLI argument parser.

Parameters **add** (*callable*) – Function that proxies calls to `argparse.ArgumentParser.add_argument` prepending options with unique plugin name prefix.

class letsencrypt.plugins.common.**Addr**(*tup*)
Bases: object

Represents an virtual host address.

Parameters

- **addr** (*str*) – addr part of vhost address
- **port** (*str*) – port number or *, or ""

classmethod fromstring(*str_addr*)
Initialize Addr from string.

get_addr()
Return addr part of Addr object.

get_port()
Return port.

get_addr_obj(*port*)
Return new address object with same addr and new port.

class letsencrypt.plugins.common.**Dvsni**(*configurator*)
Bases: object

Class that perform DVSNI challenges.

add_chall(*achall*, *idx=None*)
Add challenge to DVSNI object to perform at once.

Parameters

- **achall** (*letsencrypt.achallenges.DVSNI*) – Annotated DVSNI challenge.
- **idx** (*int*) – index to challenge in a larger array

get_cert_path(*achall*)
Returns standardized name for challenge certificate.

Parameters **achall** (*letsencrypt.achallenges.DVSNI*) – Annotated DVSNI challenge.

Returns certificate file name

Return type str

get_key_path(*achall*)
Get standardized path to challenge key.

_setup_challenge_cert(*achall*, *s=None*)
Generate and write out challenge certificate.

letsencrypt.plugins.common.**setup_ssl_options**(*config_dir*, *src*, *dest*)
Move the ssl_options into position and return the path.

letsencrypt.plugins.common.**dir_setup**(*test_dir*, *pkg*)
Setup the directories necessary for the configurator.

5.16 letsencrypt.plugins.disco

Utilities for plugins discovery and selection.

class letsencrypt.plugins.disco.**PluginEntryPoint**(*entry_point*)
Bases: object

Plugin entry point.

PREFIX_FREE_DISTRIBUTIONS = ['letsencrypt', 'letsencrypt-apache', 'letsencrypt-nginx']
Distributions for which prefix will be omitted.

classmethod entry_point_to_plugin_name(*entry_point*)
Unique plugin name for an entry_point

description
Description of the plugin.

description_with_name
Description with name. Handy for UI.

hidden
Should this plugin be hidden from UI?

ifaces(**ifaces_groups*)
Does plugin implements specified interface groups?

initialized
Has the plugin been initialized already?

init(*config=None*)
Memoized plugin inititialization.

verify(*ifaces*)
Verify that the plugin conforms to the specified interfaces.

prepared
Has the plugin been prepared already?

prepare()
Memoized plugin preparation.

misconfigured
Is plugin misconfigured?

available
Is plugin available, i.e. prepared or misconfigured?

class letsencrypt.plugins.disco.**PluginsRegistry**(*plugins*)
Bases: _abcoll.Mapping

Plugins registry.

classmethod find_all()
Find plugins using setuptools entry points.

init(*config*)
Initialize all plugins in the registry.

filter(*pred*)
Filter plugins based on predicate.

visible()
Filter plugins based on visibility.

ifaces(**ifaces_groups*)
Filter plugins based on interfaces.

verify(*ifaces*)
Filter plugins based on verification.

prepare()
Prepare all plugins in the registry.

available()
Filter plugins based on availability.

find_init(*plugin*)
Find an initialized plugin.

This is particularly useful for finding a name for the plugin (although *IPluginFactory.__call__* takes name as one of the arguments, IPlugin.name is not part of the interface):

```
        # plugin is an instance providing IPlugin, initialized
        # somewhere else in the code
        plugin_registry.find_init(plugin).name
```

Returns `None` if `plugin` is not found in the registry.

5.17 letsencrypt.plugins.manual

Manual plugin.

class letsencrypt.plugins.manual.**Authenticator**(**args*, ***kwargs*)
Bases: *letsencrypt.plugins.common.Plugin*

Manual Authenticator.

Todo

Support for `DVSNI`.

CMD_TEMPLATE = 'mkdir -p {root}/public_html/{response.URI_ROOT_PATH}\ncd {root}/public_html\necho -n {validati
Command template.

5.18 letsencrypt.plugins.standalone

Standalone Authenticator.

class letsencrypt.plugins.standalone.**ServerManager**(*certs*, *simple_http_resources*)
Bases: `object`

Standalone servers manager.

Manager for `ACMEServer` and `ACMETLSServer` instances.

`certs` and `simple_http_resources` correspond to `acme.crypto_util.SSLSocket.certs` and `acme.crypto_util.SSLSocket.simple_http_resources` respectively. All created servers share the same certificates and resources, so if you're running both TLS and non-TLS instances, SimpleHTTP handlers will serve the same URLs!

class _Instance(*server*, *thread*)
Bases: `tuple`

_asdict()
Return a new OrderedDict which maps field names to their values

classmethod _make(*iterable*, *new=<built-in method __new__ of type object at 0x9192c0>*, *len=<built-in function len>*)
Make a new _Instance object from a sequence or iterable

_replace(*_self*, ***kwds*)
Return a new _Instance object replacing specified fields with new values

server
Alias for field number 0

thread
Alias for field number 1

ServerManager.**run**(*port*, *challenge_type*)
Run ACME server on specified port.

This method is idempotent, i.e. all calls with the same pair of (port, challenge_type) will reuse the same server.

Parameters

- **port** (*int*) – Port to run the server on.
- **challenge_type** – Subclass of acme.challenges.Challenge, either acme.challenge.SimpleHTTP or acme.challenges.DVSNI.

Returns Server instance.

Return type ACMEServerMixin

ServerManager.**stop**(*port*)
Stop ACME server running on the specified port.

Parameters **port** (*int*) –

ServerManager.**running**()
Return all running instances.

Once the server is stopped using *stop*, it will not be returned.

Returns Mapping from port to server.

Return type tuple

letsencrypt.plugins.standalone.**supported_challenges_validator**(*data*)
Supported challenges validator for the argparse.

It should be passed as type argument to add_argument.

class letsencrypt.plugins.standalone.**Authenticator**(**args*, ***kwargs*)
Bases: *letsencrypt.plugins.common.Plugin*

Standalone Authenticator.

This authenticator creates its own ephemeral TCP listener on the necessary port in order to respond to incoming DVSNI and SimpleHTTP challenges from the certificate authority. Therefore, it does not rely on any existing server program.

supported_challenges
Challenges supported by this plugin.

perform2(*achalls*)
Perform achallenges without IDisplay interaction.

5.19 letsencrypt.plugins.util

Plugin utilities.

letsencrypt.plugins.util.**already_listening**(*port*)
Check if a process is already listening on the port.

If so, also tell the user via a display notification.

> **Warning:** On some operating systems, this function can only usefully be run as root.

Parameters **port** (*int*) – The TCP port in question.

Returns True or False.

5.20 letsencrypt.plugins.webroot

Webroot plugin.

class letsencrypt.plugins.webroot.**Authenticator**(**args*, ***kwargs*)
Bases: *letsencrypt.plugins.common.Plugin*

Webroot Authenticator.

5.21 letsencrypt.proof_of_possession

Proof of Possession Identifier Validation Challenge.

class letsencrypt.proof_of_possession.**ProofOfPossession**(*installer*)
Bases: object

Proof of Possession Identifier Validation Challenge.

Based on draft-barnes-acme, section 6.5.

Variables **installer** – Installer object

perform(*achall*)
Perform the Proof of Possession Challenge.

Parameters **achall** (*letsencrypt.achallenges.ProofOfPossession*) – Proof of Possession Challenge

Returns Response or None/False if the challenge cannot be completed

Return type acme.challenges.ProofOfPossessionResponse or False

_gen_response(*achall*, *key_path*)
Create the response to the Proof of Possession Challenge.

Parameters

- **achall** (*letsencrypt.achallenges.ProofOfPossession*) – Proof of Possession Challenge
- **key_path** (*str*) – Path to the key corresponding to the hinted to public key.

Returns Response or False if the challenge cannot be completed

Return type acme.challenges.ProofOfPossessionResponse or False

5.22 letsencrypt.renewer

Renewer tool.

Renewer tool handles autorenewal and autodeployment of renewed certs within lineages of successor certificates, according to configuration.

Todo

Sanity checking consistency, validity, freshness?

Todo

Call new installer API to restart servers after deployment

class letsencrypt.renewer.**_AttrDict**(**args*, ***kwargs*)
Bases: dict

Attribute dictionary.

A trick to allow accessing dictionary keys as object attributes.

letsencrypt.renewer.**renew**(*cert*, *old_version*)
Perform automated renewal of the referenced cert, if possible.

Parameters

- **cert** (letsencrypt.storage.RenewableCert) – The certificate lineage to attempt to renew.
- **old_version** (*int*) – The version of the certificate lineage relative to which the renewal should be attempted.

Returns A number referring to newly created version of this cert lineage, or `False` if renewal was not successful.

Return type `int` or `bool`

letsencrypt.renewer.**main**(*cli_args=['-b', 'latex', '-D', 'language=en', '-d', '_build/doctrees', '.', '_build/latex']*)
Main function for autorenewer script.

5.23 letsencrypt.reporter

Collects and displays information to the user.

class letsencrypt.reporter.**Reporter**
Bases: object

Collects and displays information to the user.

Variables **messages** (*Queue.PriorityQueue*) – Messages to be displayed to the user.

HIGH_PRIORITY = 0
High priority constant. See *add_message*.

MEDIUM_PRIORITY = 1
Medium priority constant. See *add_message*.

LOW_PRIORITY = 2
Low priority constant. See *add_message*.

_msg_type
alias of `ReporterMsg`

add_message(*msg*, *priority*, *on_crash=True*)
Adds msg to the list of messages to be printed.

Parameters

- **msg** (*str*) – Message to be displayed to the user.

- **priority** (*int*) – One of *HIGH_PRIORITY*, *MEDIUM_PRIORITY*, or *LOW_PRIORITY*.
- **on_crash** (*bool*) – Whether or not the message should be printed if the program exits abnormally.

atexit_print_messages(*pid=727*)
Function to be registered with atexit to print messages.

Parameters **pid** (*int*) – Process ID

print_messages()
Prints messages to the user and clears the message queue.

If there is an unhandled exception, only messages for which `on_crash` is `True` are printed.

5.24 letsencrypt.reverter

Reverter class saves configuration checkpoints and allows for recovery.

class letsencrypt.reverter.**Reverter**(*config*)
Bases: object

Reverter Class - save and revert configuration checkpoints.

Note: Consider moving everything over to CSV format.

Parameters **config** (*letsencrypt.interfaces.IConfig*) – Configuration.

revert_temporary_config()
Reload users original configuration files after a temporary save.

This function should reinstall the users original configuration files for all saves with temporary=True

Raises .ReverterError when unable to revert config

rollback_checkpoints(*rollback=1*)
Revert 'rollback' number of configuration checkpoints.

Parameters **rollback** (*int*) – Number of checkpoints to reverse. A str num will be cast to an integer. So "2" is also acceptable.

Raises .ReverterError if there is a problem with the input or if the function is unable to correctly revert the configuration checkpoints

view_config_changes()
Displays all saved checkpoints.

All checkpoints are printed by *letsencrypt.interfaces.IDisplay.notification()*.

Todo

Decide on a policy for error handling, OSError IOError...

Raises .errors.ReverterError If invalid directory structure.

add_to_temp_checkpoint(*save_files*, *save_notes*)
Add files to temporary checkpoint.

Parameters

- **save_files** (*set*) – set of filepaths to save
- **save_notes** (*str*) – notes about changes during the save

add_to_checkpoint(*save_files*, *save_notes*)
Add files to a permanent checkpoint.

Parameters

- **save_files** (*set*) – set of filepaths to save
- **save_notes** (*str*) – notes about changes during the save

_add_to_checkpoint_dir(*cp_dir*, *save_files*, *save_notes*)
Add save files to checkpoint directory.

Parameters

- **cp_dir** (*str*) – Checkpoint directory filepath
- **save_files** (*set*) – set of files to save
- **save_notes** (*str*) – notes about changes made during the save

Raises

- **IOError** – if unable to open cp_dir + FILEPATHS file
- **.ReverterError** – if unable to add checkpoint

_read_and_append(*filepath*)
Reads the file lines and returns a file obj.

Read the file returning the lines, and a pointer to the end of the file.

_recover_checkpoint(*cp_dir*)
Recover a specific checkpoint.

Recover a specific checkpoint provided by cp_dir Note: this function does not reload augeas.

Parameters **cp_dir** (*str*) – checkpoint directory file path

Raises errors.ReverterError If unable to recover checkpoint

_run_undo_commands(*filepath*)
Run all commands in a file.

_check_tempfile_saves(*save_files*)
Verify save isn't overwriting any temporary files.

Parameters **save_files** (*set*) – Set of files about to be saved.

Raises *letsencrypt.errors.ReverterError* when save is attempting to overwrite a temporary file.

register_file_creation(*temporary*, **files*)
Register the creation of all files during letsencrypt execution.

Call this method before writing to the file to make sure that the file will be cleaned up if the program exits unexpectedly. (Before a save occurs)

Parameters

- **temporary** (*bool*) – If the file creation registry is for a temp or permanent save.
- ***files** – file paths (str) to be registered

Raises *letsencrypt.errors.ReverterError* If call does not contain necessary parameters or if the file creation is unable to be registered.

register_undo_command(*temporary*, *command*)
Register a command to be run to undo actions taken.

> **Warning:** This function does not enforce order of operations in terms of file modification vs. command registration. All undo commands are run first before all normal files are reverted to their previous state. If you need to maintain strict order, you may create checkpoints before and after the the command registration. This function may be improved in the future based on demand.

Parameters

- **temporary** (*bool*) – Whether the command should be saved in the IN_PROGRESS or TEMPORARY checkpoints.
- **command** (*list of str*) – Command to be run.

_get_cp_dir(*temporary*)
Return the proper reverter directory.

recovery_routine()
Revert configuration to most recent finalized checkpoint.

Remove all changes (temporary and permanent) that have not been finalized. This is useful to protect against crashes and other execution interruptions.

Raises .errors.ReverterError If unable to recover the configuration

_remove_contained_files(*file_list*)
Erase all files contained within file_list.

Parameters **file_list** (*str*) – file containing list of file paths to be deleted

Returns Success

Return type bool

Raises *letsencrypt.errors.ReverterError* If all files within file_list cannot be removed

finalize_checkpoint(*title*)
Finalize the checkpoint.

Timestamps and permanently saves all changes made through the use of *add_to_checkpoint()* and *register_file_creation()*

Parameters **title** (*str*) – Title describing checkpoint

Raises *letsencrypt.errors.ReverterError* when the checkpoint is not able to be finalized.

_timestamp_progress_dir()
Timestamp the checkpoint.

5.25 letsencrypt.storage

Renewable certificates storage.

letsencrypt.storage.**config_with_defaults**(*config=None*)
Merge supplied config, if provided, on top of builtin defaults.

letsencrypt.storage.**parse_time_interval**(*interval*, *textparser=<parsedatetime.Calendar instance>*)

Parse the time specified time interval.

The interval can be in the English-language format understood by parsedatetime, e.g., '10 days', '3 weeks', '6 months', '9 hours', or a sequence of such intervals like '6 months 1 week' or '3 days 12 hours'. If an integer is found with no associated unit, it is interpreted by default as a number of days.

Parameters **interval** (*str*) – The time interval to parse.

Returns The interpretation of the time interval.

Return type datetime.timedelta

class letsencrypt.storage.**RenewableCert**(*config_filename*, *cli_config*)

Bases: object

Renewable certificate.

Represents a lineage of certificates that is under the management of the Let's Encrypt client, indicated by the existence of an associated renewal configuration file.

Note that the notion of "current version" for a lineage is maintained on disk in the structure of symbolic links, and is not explicitly stored in any instance variable in this object. The RenewableCert object is able to determine information about the current (or other) version by accessing data on disk, but does not inherently know any of this information except by examining the symbolic links as needed. The instance variables mentioned below point to symlinks that reflect the notion of "current version" of each managed object, and it is these paths that should be used when configuring servers to use the certificate managed in a lineage. These paths are normally within the "live" directory, and their symlink targets – the actual cert files – are normally found within the "archive" directory.

Variables

- **cert** (*str*) – The path to the symlink representing the current version of the certificate managed by this lineage.
- **privkey** (*str*) – The path to the symlink representing the current version of the private key managed by this lineage.
- **chain** (*str*) – The path to the symlink representing the current version of the chain managed by this lineage.
- **fullchain** (*str*) – The path to the symlink representing the current version of the fullchain (combined chain and cert) managed by this lineage.
- ***configuration*** (*configobj.ConfigObj*) – The renewal configuration options associated with this lineage, obtained from parsing the renewal configuration file and/or systemwide defaults.

_consistent()

Are the files associated with this lineage self-consistent?

Returns Whether the files stored in connection with this lineage appear to be correct and consistent with one another.

Return type bool

_fix()

Attempt to fix defects or inconsistencies in this lineage.

Todo

Currently unimplemented.

_previous_symlinks()
Returns the kind and path of all symlinks used in recovery.

Returns list of (kind, symlink) tuples

Return type list

_fix_symlinks()
Fixes symlinks in the event of an incomplete version update.

If there is no problem with the current symlinks, this function has no effect.

current_target(*kind*)
Returns full path to which the specified item currently points.

Parameters **kind** (*str*) – the lineage member item ("cert", "privkey", "chain", or "fullchain")

Returns The path to the current version of the specified member.

Return type str

current_version(*kind*)
Returns numerical version of the specified item.

For example, if kind is "chain" and the current chain link points to a file named "chain7.pem", returns the integer 7.

Parameters **kind** (*str*) – the lineage member item ("cert", "privkey", "chain", or "fullchain")

Returns the current version of the specified member.

Return type int

version(*kind*, *version*)
The filename that corresponds to the specified version and kind.

Warning: The specified version may not exist in this lineage. There is no guarantee that the file path returned by this method actually exists.

Parameters

- **kind** (*str*) – the lineage member item ("cert", "privkey", "chain", or "fullchain")
- **version** (*int*) – the desired version

Returns The path to the specified version of the specified member.

Return type str

available_versions(*kind*)
Which alternative versions of the specified kind of item exist?

The archive directory where the current version is stored is consulted to obtain the list of alternatives.

Parameters **kind** (*str*) – the lineage member item (`cert`, `privkey`, `chain`, or `fullchain`)

Returns all of the version numbers that currently exist

Return type `list` of `int`

newest_available_version(*kind*)
Newest available version of the specified kind of item?

Parameters **kind** (*str*) – the lineage member item (`cert`, `privkey`, `chain`, or `fullchain`)

Returns the newest available version of this member

Return type int

latest_common_version()
Newest version for which all items are available?

Returns the newest available version for which all members (`cert`, `privkey`, `chain`, and `fullchain`) exist

Return type int

next_free_version()
Smallest version newer than all full or partial versions?

Returns the smallest version number that is larger than any version of any item currently stored in this lineage

Return type int

has_pending_deployment()
Is there a later version of all of the managed items?

Returns `True` if there is a complete version of this lineage with a larger version number than the current version, and `False` otherwis

Return type bool

_update_link_to(*kind*, *version*)
Make the specified item point at the specified version.

(Note that this method doesn't verify that the specified version exists.)

Parameters

- **kind** (*str*) – the lineage member item ("cert", "privkey", "chain", or "fullchain")
- **version** (*int*) – the desired version

update_all_links_to(*version*)
Change all member objects to point to the specified version.

Parameters **version** (*int*) – the desired version

names(*version=None*)
What are the subject names of this certificate?

(If no version is specified, use the current version.)

Parameters **version** (*int*) – the desired version number

Returns the subject names

Return type `list` of `str`

autodeployment_is_enabled()
Is automatic deployment enabled for this cert?

If autodeploy is not specified, defaults to True.

Returns True if automatic deployment is enabled

Return type bool

should_autodeploy()
Should this lineage now automatically deploy a newer version?

This is a policy question and does not only depend on whether there is a newer version of the cert. (This considers whether autodeployment is enabled, whether a relevant newer version exists, and whether the time interval for autodeployment has been reached.)

Returns whether the lineage now ought to autodeploy an existing newer cert version

Return type bool

ocsp_revoked(*version=None*)

Is the specified cert version revoked according to OCSP?

Also returns True if the cert version is declared as intended to be revoked according to Let's Encrypt OCSP extensions. (If no version is specified, uses the current version.)

This method is not yet implemented and currently always returns False.

Parameters **version** (*int*) – the desired version number

Returns whether the certificate is or will be revoked

Return type bool

autorenewal_is_enabled()

Is automatic renewal enabled for this cert?

If autorenew is not specified, defaults to True.

Returns True if automatic renewal is enabled

Return type bool

should_autorenew()

Should we now try to autorenew the most recent cert version?

This is a policy question and does not only depend on whether the cert is expired. (This considers whether autorenewal is enabled, whether the cert is revoked, and whether the time interval for autorenewal has been reached.)

Note that this examines the numerically most recent cert version, not the currently deployed version.

Returns whether an attempt should now be made to autorenew the most current cert version in this lineage

Return type bool

classmethod new_lineage(*lineagename*, *cert*, *privkey*, *chain*, *renewalparams=None*, *config=None*, *cli_config=None*)

Create a new certificate lineage.

Attempts to create a certificate lineage – enrolled for potential future renewal – with the (suggested) lineage name lineagename, and the associated cert, privkey, and chain (the associated fullchain will be created automatically). Optional configurator and renewalparams record the configuration that was originally used to obtain this cert, so that it can be reused later during automated renewal.

Returns a new RenewableCert object referring to the created lineage. (The actual lineage name, as well as all the relevant file paths, will be available within this object.)

Parameters

- **lineagename** (*str*) – the suggested name for this lineage (normally the current cert's first subject DNS name)
- **cert** (*str*) – the initial certificate version in PEM format
- **privkey** (*str*) – the private key in PEM format
- **chain** (*str*) – the certificate chain in PEM format

- **renewalparams** (*configobj.ConfigObj*) – parameters that should be used when instantiating authenticator and installer objects in the future to attempt to renew this cert or deploy new versions of it
- **config** (*configobj.ConfigObj*) – renewal configuration defaults, affecting, for example, the locations of the directories where the associated files will be saved
- **cli_config** (RenewerConfiguration) – parsed command line arguments

Returns the newly-created RenewalCert object

Return type `storage.renewableCert`

save_successor(*prior_version*, *new_cert*, *new_privkey*, *new_chain*)

Save new cert and chain as a successor of a prior version.

Returns the new version number that was created.

Note: this function does NOT update links to deploy this version

---

Parameters

- **prior_version** (*int*) – the old version to which this version is regarded as a successor (used to choose a privkey, if the key has not changed, but otherwise this information is not permanently recorded anywhere)
- **new_cert** (*str*) – the new certificate, in PEM format
- **new_privkey** (*str*) – the new private key, in PEM format, or `None`, if the private key has not changed
- **new_chain** (*str*) – the new chain, in PEM format

Returns the new version number that was created

Return type int

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

l

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Y