django-ca Documentation

Release 1.4.1

Mathias Ertl

February 26, 2017

Contents

django-ca is a small project to manage TLS certificate authorities and easily issue and revoke certificates. It is based on pyOpenSSL and Django. It can be used as an app in an existing Django project or stand-alone with the basic project included. Certificates can be managed through Djangos admin interface or via *manage.py* commands - so no webserver is needed, if you're happy with the command-line.

Features:

- Create certificate authorities, issue and revoke certificates in minutes.
- Receive e-mail notifications of certificates about to expire.
- Certificate validation via the included OCSP responder and Certificate Revocation Lists (CRLs).
- Complete, consistent and powerful command line interface.
- Optional web interface for certificate handling (e.g. issuing, revoking, ...).
- Written in pure Python2.7/Python3.4+, using Django 1.8 or later.

Contents:

CHAPTER 1

Installation

You can run **django-ca** as a regular app in any existing Django project of yours, but if you don't have any Django project running, you can run it as a *standalone project*.

Requirements

- Python 2.7 or Python 3.4+
- Django 1.8+
- Any database supported by Django (sqlite3/MySQL/PostgreSQL/...)
- Python, OpenSSL and libffi development headers

As Django app (in your existing Django project)

This chapter assumes that you have an already running Django project and know how to use it.

You need various development headers for pyOpenSSL, on Debian/Ubuntu systems, simply install these packages:

```
$ apt-get install gcc python3-dev libffi-dev libssl-dev
```

You can install **django-ca** simply via pip:

```
$ pip install django-ca
```

and add it to your `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    # ... your other apps...

    'django_ca',
]
```

... and configure the other available settings to your liking, then simply run:

```
$ python manage.py migrate
$ python manage.py collectstatic

# FINALLY, create the root certificates for your CA:
#      (replace parameters after init_ca with your local details)
```

```
$ python manage.py init_ca RootCA \
>     /C=AT/ST=Vienna/L=Vienna/O=Org/OU=OrgUnit/CN=ca.example.com
```

After that, **django-ca** should show up in your admin interface (see Web interface) and provide various `manage.py` commands (see Command-line interface).

As standalone project

You can also install **django-ca** as a stand-alone project, if you install it via git. The project provides a command-line interface that provides complete functionality. The web interface is optional.

Note: If you don't want the private keys of your CAs on the same machine as the web interface, you can also host the web interface on a second server that accesses the same database (CA private keys are hosted on the filesystem, not in the database). You obviously will not be able to sign certificates using the web interface, but you can still e.g. revoke certificates or run a OCSP responder.

In the following code-snippet, you'll do all necessary steps to get a basic setup:

```
# install dependencies (adapt to your distro):
$ apt-get install gcc git python3-dev libffi-dev libssl-dev virtualenv

# clone git repository:
$ git clone https://github.com/mathiasertl/django-ca.git

# create virtualenv:
$ cd django-ca
$ virtualenv -p /usr/bin/python3 .
$ source bin/activate

# install Python dependencies:
$ pip install -U pip setuptools
$ pip install -r requirements.txt
```

In the above script, you have created a virtualenv, meaning that all libraries you install with `pip install` are installed in the virtualenv (and don't pollute your system). It also means that before you execute any `manage.py` commands, you'll have to activate your virtualenv, by doing, in the directory of the git checkout:

```
$ source bin/activate
```

Configure django-ca

Before you continue, you have to configure **django-ca**. Django uses a file called `settings.py`, but so you don't have to change any files managed by git, it includes `localsettings.py` in the same directory. So copy the example file and edit it with your favourite editor:

```
$ cp ca/ca/localsettings.py.example ca/ca/localsettings.py
```

The most important settings are documented there, but you can of course use any setting provided by Django.

> **Warning:** The `SECRET_KEY` and `DATABASES` settings are absolutely mandatory. If you use the Web interface, the `STATIC_ROOT` setting is also mandatory.

Initialize the project

After you have configured **django-ca**, you need to initialize the project by running a few `manage.py` commands:

```
$ python ca/manage.py migrate

# If you intend to run the webinterface (requires STATIC_ROOT setting!)
$ python ca/manage.py collectstatic

# FINALLY, create a certificate authority:
#     (replace parameters after init_ca with your local details)
$ python manage.py init_ca /C=AT/ST=Vienna/L=Vienna/O=Org/CN=ca.example.com
```

Please also see Certificate authority management for further information on how to create certificate authorities. You can also run `init_ca` with the `-h` parameter for available arguments.

Create manage.py shortcut

If you don't want to always chdir to the git checkout, activate the virtualenv and only then run `manage.py`, you might want to create a shortcut shell script somewhere in your `PATH` (e.g. `/usr/local/bin`):

```
#!/bin/bash

# BASEDIR is the location of your git checkout
BASEDIR=/usr/local/share/ca
PYTHON=${BASEDIR}/bin/python
MANAGE=${BASEDIR}/ca/manage.py

${PYTHON} ${MANAGE} "$@"
```

Setup a webserver

Setting up a webserver and all that comes with it is really out of scope of this document. The WSGI file is located in `ca/ca/wsgi.py`. Django itself provides some info for using Apache and mod_wsgi, or you could use uWSGI and nginx, or any of the many other options available.

Apache and mod_wsgi

Github user Raoul Thill notes that you need some special configuration variable if you use Apache together with mod_wsgi (see here):

```
WSGIDaemonProcess django_ca processes=1 python-path=/opt/django-ca/ca:/opt/django-ca/ca/ca:/opt/djan
WSGIProcessGroup django_ca
WSGIApplicationGroup %{GLOBAL}
WSGIScriptAlias / /opt/django-ca/ca/ca/wsgi.py
```

Regular cronjobs

Some `manage.py` commands are intended to be run as cronjobs:

```
# assuming you cloned the repo at /root/:
HOME=/root/django-ca
PATH=/root/django-ca/bin

# m h  dom mon dow      user  command

# notify watchers about certificates about to expire
* 8    * * *            root  python ca/manage.py notify_expiring_certs

# recreate the CRL and the OCSP index
12 *    * * *           root  python ca/manage.py dump_crl
14 *    * * *           root  python ca/manage.py dump_ocsp_index
```

CHAPTER 2

Update

Since 1.0.0, this project updates like any other project. First, update the source code, if you use git:

```
git pull origin master
```

or if you installed **django-ca** via pip:

```
pip install -U django-ca
```

then upgrade with these commands:

```
pip install -U -r requirements.txt
python ca/manage.py migrate

# if you use the webinterface
python ca/manage.py collectstatic
```

Warning: If you installed **django-ca** in a virtualenv, don't forget to activate it before executing any python or pip commands using:

```
source bin/activate
```

Update from 1.0.0b2

If you're updating from a version earlier then 1.0.0 (which was the first real release), you have to first update to 1.0.0.b1 (see below), then to 1.0.0.b2, apply all migrations and reset existing migrations Since all installed instances were probably private, it made sense to start with a clean state.

To update from an earlier git-checkout, to:

- Upgrade to version 1.0.0b2
- Apply all migrations.
- Upgrade to version 1.0.0
- Remove old migrations from the database:

```
  python manage.py dbshell
  > DELETE FROM django_migrations WHERE app='django_ca';
```

- Fake the first migration:

  python manage.py migrate django_ca 0001 –fake

Update from pre 1.0.0b1

Prior to 1.0.0, this app was not intended to be reusable and so had a generic name. The app was renamed to *django_ca*, so it can be used in other Django projects (or hopefully stand-alone, someday). Essentially, the upgrade path should work something like this:

```
# backup old data:
python manage.py dumpdata certificate --indent=4 > certs.json

# update source code
git pull origin master

# create initial models in the new app, but only the initial version!
python manage.py migrate django_ca 0001

# update JSON with new model name
sed 's/"certificate.certificate"/"django_ca.certificate"/' > certs-updated.json

# load data
python manage.py loaddata certs-updated.json

# apply any other migrations
python manage.py migrate
```

CHAPTER 3

Custom settings

You can use any of the settings understood by Django and **django-ca** provides some of its own settings.

From Djangos settings, you especially need to configure `DATABASES`, `SECRET_KEY`, `ALLOWED_HOSTS` and `STATIC_ROOT`.

All settings used by **django-ca** start with the `CA_` prefix. Settings are also documented at `ca/ca/localsettings.py.example` (view on git).

CA_DEFAULT_EXPIRES Default: `730`

The default time, in days, that any signed certificate expires.

CA_DEFAULT_PROFILE Default: `webserver`

The default profile to use.

CA_DEFAULT_SUBJECT Default: `{}`

The default subject to use. The keys of this dictionary are the valid fields in X509 certificate subjects. Example:

```
CA_DEFAULT_SUBJECT = {
    'C': 'AT',
    'ST': 'Vienna',
    'L': 'Vienna',
    'O': 'HTU Wien',
    'OU': 'Fachschaft Informatik',
    'emailAddress': 'user@example.com',
}
```

CA_DIGEST_ALGORITHM Default: `"sha512"`

The default digest algorithm used to sign certificates. You may want to use `"sha256"` for older (pre-2010) clients. Note that this setting is also used by the `init_ca` command, so if you have any clients that do not understand sha512 hashes, you should change this beforehand.

CA_DIR Default: `"ca/files"`

Where the root certificate is stored. The default is a `files` directory in the same location as your `manage.py` file.

CA_NOTIFICATION_DAYS Default: `[14, 7, 3, 1, ]`

Days before expiry that certificate watchers will receive notifications. By default, watchers will receive notifications 14, seven, three and one days before expiry.

CA_OCSP_URLS Default: `{}`

Configuration for OCSP responders. See Run a OCSP responder for more information.

CA_PROFILES Default: `{}`

Profiles determine the default values for the `keyUsage`, `extendedKeyUsage` x509 extensions. In short, they determine how your certificate can be used, be it for server and/or client authentication, e-mail signing or anything else. By default, **django-ca** provides these profiles:

Profile	keyUsage	extendedKeyUsage
client	digitalSignature	clientAuth
server	digitalSignature, keyAgreement keyEncipherment	clientAuth, serverAuth
web-server	digitalSignature, keyAgreement keyEncipherment	serverAuth
enduser	dataEncipherment, digitalSignature, keyEncipherment	clientAuth, emailProtection, codeSigning
ocsp	nonRepudiation, talSignature, keyEncipherment	OCSPSigning

Further more,

- The `keyUsage` attribute is marked as critical.
- The `extendedKeyUsage` attribute is marked as non-critical.

This should be fine for most usecases. But you can use the `CA_PROFILES` setting to either update or disable existing profiles or add new profiles that you like. For that, set `CA_PROFILES` to a dictionary with the keys defining the profile name and the value being either:

- `None` to disable an existing profile.
- A dictionary defining the profile. If the name of the profile is an existing profile, the dictionary is updated, so you can ommit a value to leave it as the default. The possible keys are:

key	Description
`"keyUsage"`	The `keyUsage` X509 extension.
`"extendedKeyUsage"`	The `extendedKeyUsage` X509 extension.
`"desc"`	A human-readable description, shows up with “sing_cert -h” and in the webinterface profile selection.
`"subject"`	The default subject to use. If ommited, `CA_DEFAULT_SUBJECT` is used.
`"cn_in_san"`	If to include the CommonName in the subjectAltName by default. The default value is `True`.

Here is a full example:

```
CA_DEFAULT_PROFILES = {
    'client': {
        'desc': _('Nice description.'),
        'keyUsage': {
            'critical': True,
            'value': [
               'digitalSignature',
            ],
        },
        'extendedKeyUsage': {
            'critical': False,
            'value': [
               'clientAuth',
            ],
         },
         'subject': {
            'C': 'AT',
            'L': 'Vienna',
         }
```

```
            },

            # We really don't like the "ocsp" profile, so we remove it.
            'ocsp': None,
        }
```

CA_PROVIDE_GENERIC_CRL Default: `True`

If set to `False`, `django_ca.urls` will not add a CRL view. See *Use generic view to host a CRL* for more information.

This setting only has effect if you use django_ca as a full project or you include the `django_ca.urls` module somewhere in your URL configuration.

CHAPTER 4

Command-line interface

django-ca provides a complete command-line interface for all functionality. It is implemented as subcommands of Djangos `manage.py` script. You can use it for all certificate management operations, and Certificate authority management is only possible via the command-line interface for security reasons.

In general, run `manage.py` without any parameters for available subcommands:

```
$ python manage.py

...
[django_ca]
    cert_watchers
    dump_cert
    dump_crl
    ...
```

Warning: Remember to use the virtualenv if you installed **django-ca** in one.

Execute `manage.py <subcommand> -h` to get help on the subcommand.

`manage.py` subcommands for certificate authority management:

Command	Description
dump_ca	Write the CA certificate to a file.
edit_ca	Edit an existing certificate authority.
init_ca	Create a new certificate authority.
list_cas	List currently configured certificate authorities.
view_ca	View details of a certificate authority.

`manage.py` subcommands for certificate management:

Command	Description
cert_watchers	Add/remove addresses to be notified of an expiring certificate.
dump_cert	Dump a certificate to a file.
list_certs	List all certificates.
notify_expiring_certs	Send notifications about expiring certificates to watchers.
revoke_cert	Revoke a certificate.
sign_cert	Sign a certificate.
view_cert	View a certificate.

Miscellaneous `manage.py` subcommands:

Command	Description
dump_crl	Write the certificate revocation list (CRL), see Host a Certificate Revokation List (CRL).
dump_ocsp_index	Write an OCSP index file, see Run a OCSP responder.

CHAPTER 5

Web interface

The web interface allows you to perform the most common tasks necessary when running certificate authority. It is implemented using Djangos admin interface. You can:

- Issue and revoke certificates.
- Modify the x509 extensions used when signing certificates.
- Modify who is notified about expiring certificates.

The django project in the git repository (e.g. if you installed **django-ca** as *a standalone project*) already enables the admin interface and it's usable as soon as you enabled the webserver (tip: Create a user for login using `manage.py createsuperuser`). If you installed **django-ca** as an app, the admin interface is automatically included.

CHAPTER 6

Certificate authority management

django-ca supports managing multiple certificate authorities as well as child certificate authorities.

The only way to create certificate authorities is via the command-line interface. It is obviously most important that the private keys of the certificate authorities are never exposed to any attacker, and any web interface would pose an unnecessary risk.

For the same reason, the private key of a certificate authority is stored on the filesystem and not in the database. The initial location of the private key is configured by the *CA_DIR setting*. This also means that you can run your **django-ca** on two hosts, where one host has the private key and only uses the command line, and one with the webinterface that can still be used to revoke certificates.

To manage certificate authorities, use the following *manage.py* commands:

Command	Description
init_ca	Create a new certificate authority.
list_cas	List all currently configured certificate authorities.
edit_ca	Edit a certificate authority.
view_ca	View details of a certificate authority.
dump_ca	Write the CA certificate to a file.

Various details of the certificate authority, mostly the x509 extensions used when signing a certificate, can also be managed via the webinterface.

Here is a shell session that illustrates the respective manage.py commands:

```
$ python manage.py init_ca --pathlen=2
>     --crl-url=http://ca.example.com/crl \
>     --ocsp-url=http://ocsp.ca.example.com \
>     --issuer-url=http://ca.example.com/ca.crt \
>     TestCA /C=AT/L=Vienna/L=Vienna/O=Example/OU=ExampleUnit/CN=ca.example.com
$ python manage.py list_cas
BD:5B:AB:5B:A2:1C:49:0D:9A:B2:AA:BC:68:ED:ED:7D - TestCA

$ python manage.py view_ca BD:5B:AB:5B:A2
...
* OCSP URL: http://ocsp.ca.example.com
$ python manage.py edit_ca --ocsp-url=http://new-ocsp.ca.example.com \
>     BD:5B:AB:5B:A2
$ python manage.py view_ca BD:5B:AB:5B:A2
...
* OCSP URL: http://new-ocsp.ca.example.com
```

Note that you can just use the start of a serial to identify the CA, as long as that still uniquely identifies the CA.

Create intermediate CAs

Intermediate CAs are created, just like normal CAs, using `manage.py init_ca`. For intermediate CAs to be valid, CAs however must have a correct `pathlen` x509 extension. Its value is an integer describing how many levels of intermediate CAs a CA may have. A `pathlen` of "0" means that a CA cannot have any intermediate CAs, if it is not present, a CA may have an infinite number of intermediate CAs.

Note: **django-ca** by default sets a `pathlen` of "0", as it aims to be secure by default. The `pathlen` attribute cannot be changed in hindsight (not without resigning the CA). If you plan to create intermediate CAs, you have to consider this when creating the root CA.

So for example, if you want two levels of intermediate CAs, , you'd need the following `pathlen` values (the `pathlen` value is the minimum value, it could always be a larger number):

index	CA	pathlen	description
1	example.com	2	Your root CA.
2	sub1.example.com	1	Your first intermediate CA, a sub-CA from (1).
3	sub2.example.com	0	A second intermediate CA, also a sub-CA from (1).
4	sub.sub1.example.com	0	An intermediate CA of (2).

If in the above example, CA (1) had `pathlen` of "1" or CA (2) had a `pathlen` of "0", CA (4) would no longer be a valid CA.

By default, **django-ca** sets a `pathlen` of 0, so CAs will not be able to have any intermediate CAs. You can configure the value by passing `--pathlen` to `init_ca`:

```
$ python manage.py init_ca --pathlen=2 ...
```

When creating a sub-ca, you must name its parent using the `--parent` parameter:

```
$ python manage.py list_cas
BD:5B:AB:5B:A2:1C:49:0D:9A:B2:AA:BC:68:ED:ED:7D - Root CA
$ python manage.py init_ca --parent=BD:5B:AB:5B ...
```

Note: Just like throughout the system, you can always just give the start of the serial, as long as it still is a unique identifier for the CA.

CHAPTER 7

Host a Certificate Revokation List (CRL)

A Certificate Revokation List (CRL) contains all revoked certificates signed by a certificate authority. Having a CRL is completely optional (e.g. Let's Encrypt certificates don't have one).

A URL to the CRL is usually included in the certificates (in the `crlDistributionPoints` x509 extension) so clients can fetch the CRL and verify that the certificate has not been revoked. Some services (e.g. OpenVPN) also just keep a local copy of a CRL.

Note: CRLs are usually hosted via HTTP, **not** HTTPS. CRLs are always signed, so hosting them via HTTP is not a security vulnerability. On the other hand, you cannot verify the the certificate used when fetching the CRL anyway, since you would need the CRL for that.

Add CRL URL to new certificates

To include the URL to a CRL in newly issued certificates (you cannot add it to already issued certificates, obviously), either set it in the admin interface or via the command line:

```
$ python manage.py list_cas
34:D6:02:B5:B8:27:4F:51:9A:16:0C:B8:56:B7:79:3F - Root CA
$ python manage.py edit_ca --crl-url=http://ca.example.com/crl.pem \
>     34:D6:02:B5:B8:27:4F:51:9A:16:0C:B8:56:B7:79:3F
```

Use generic view to host a CRL

django-ca provides the generic view *`CertificateRevocationListView`* to provide CRLs via HTTP.

If you installed **django-ca** as a full project, a default CRL is already available for all CAs. If you installed django-ca on "ca.example.com", the CRL is available at `http://ca.example.com/django_ca/crl/<serial>/`. If you installed django-ca as an app, you only need to include `django_ca.urls` in your URL conf at the appropriate location.

The default CRL is in the ASN1/DER format, signed with sha512 and refreshed every ten minutes. This is fine for TLS clients that use CRLs and is in fact similar to what public CAs use (see *crlDistributionPoints*). If you want to change any of these settings, you can override them as parameters in a URL conf:

```
from OpenSSL import crypto
from django_ca.views import CertificateRevocationListView
```

```
urlpatterns = [
   # ... your other patterns

   # We need a CRL in PEM format with a sha256 digest
   url(r'^crl/(?P<serial>[0-9A-F:]+)/$',
       CertificateRevocationListView.as_view(
           type=crypto.FILETYPE_PEM,
           digest='sha256',
           content_type='text/plain',
       ),
       name='sha256-crl')),
]
```

If you do not want to include the automatically hosted CRL, please set `CA_PROVIDE_GENERIC_CRL` to `False` in your settings.

class `django_ca.views.`**`CertificateRevocationListView`**(***kwargs*)

Generic view that provides Certificate Revocation Lists (CRLs).

`content_type` = **'application/pkix-crl'**

The value of the Content-Type header used in the response. For CRLs in PEM format, use `"text/plain"`.

`digest` = **'sha512'**

Digest used for generating the CRL.

`expires` = **600**

CRL expires in this many seconds.

`type` = **2**

Filetype for CRL, one of the `OpenSSL.crypto.FILETYPE_*` variables. The default is `OpenSSL.crypto.FILETYPE_ASN1`.

Write a CRL to a file

You can generate the CRL with the `manage.py dump_crl` command:

```
$ python manage.py dump_crl -f PEM /var/www/crl.pem
```

Note: The `dump_crl` command uses the first enabled CA by default, you can force a particular CA with `--ca=<serial>`.

CRLs expire after a certain time (default: one day, configure with `--expires=SECS`), so you must periodically regenerate it, e.g. via a cron-job.

How and where to host the file is entirely up to you. If you run a Django project with a webserver already, one possibility is to dump it to your `MEDIA_ROOT` directory.

CHAPTER 8

Run a OCSP responder

OCSP, or the Online Certificate Status Protocol provides a second method (besides CRLs) for a client to find out if a certificate has been revoked.

> **Warning:** The OCSP responder included in **django-ca** is still very experimental. Expect problems when using it. Please also expect major changes in how it is configured in future versions.

Configure OCSP with django-ca

django-ca provides generic HTTP endpoints for an OCSP service for your certificate authorities. The setup involves:

1. *Creating a responder certificate*
2. *Configure generic views*
3. *Add a OCSP URL to the new certificate*

New in version 1.2: Before version 1.2, django-ca was not able to host its own OCSP responder.

Create an OCSP responser certificate

To run an OCSP responder, you first need a certificate with some special properties. Luckily, **django-ca** has a profile predefined for you:

```
$ openssl genrsa -out ocsp.key 4096
$ openssl req -new -key ocsp.key -out ocsp.csr -utf8 -batch
$ python manage.py sign_cert --csr=ocsp.csr --out=ocsp.pem \
>     --subject /CN=ocsp.example.com --ocsp
```

> **Warning:** The CommonName in the certificates subject must match the domain where you host your **django-ca** installation.

Configure generic views

The final step in configuring an OCSP responder for the CA is configuring the HTTP endpoint. If you've installed django-ca as a full project or include `django_ca.urls` in your root URL config, configure the `CA_OCSP_URLS` setting. It's a dictionary configuring instances of *`OCSPView`*. Keys become part of the URL pattern, the value is a dictionary for the arguments of the view. For example:

```
CA_OCSP_URLS = {
    'root': {
        'ca': '34:D6:02:B5:B8:27:4F:51:9A:16:0C:B8:56:B7:79:3F',
        'responder_key': '/usr/share/django-ca/ocsp.key',
        'responder_cert': 'F2:5F:7F:31:E1:91:4F:D7:9A:D4:19:65:17:3D:43:88',
        # optional: How long OCSP responses are valid
        #'expires': 3600,
    },
}
```

This would mean that your OCSP responder would be located at `/django_ca/ocsp/root/` at whatever domain you have configured your WSGI daemon. If you're using your own URL configuration, pass the same parameters to the `as_view()` method.

class `django_ca.views.`**`OCSPView`**(***kwargs*)

View to provide an OCSP responder.

See also:

This is heavily inspired by https://github.com/threema-ch/ocspresponder/blob/master/ocspresponder/__init__.py.

`ca` = **None**

The serial of your certificate authority.

`expires` = **600**

Time in seconds that the responses remain valid. The default is 600 seconds or ten minutes.

`responder_cert` = **None**

Absolute path, serial of the public key or key itself used for signing OCSP responses.

`responder_key` = **None**

Absolute path to the private key used for signing OCSP responses.

Add OCSP URL to new certificates

To include the URL to an OCSP service to newly issued certificates (you cannot add it to already issued certificates, obviously), either set it in the admin interface or via the command line:

```
$ python manage.py list_cas
34:D6:02:B5:B8:27:4F:51:9A:16:0C:B8:56:B7:79:3F - Root CA
$ python manage.py edit_ca --ocsp-url=http://ocsp.example.com/ \
>     34:D6:02:B5:B8:27:4F:51:9A:16:0C:B8:56:B7:79:3F
```

Run an OCSP responser with `openssl ocsp`

OpenSSL ships with the `openssl ocsp` command that allows you to run an OCSP responser, but note that the manpage says **"only useful for test and demonstration purposes"**.

To use the command, generate an index:

```
$ python manage.py dump_ocsp_index ocsp.index
```

OpenSSL itself allows you to run an OCSP responder with this command:

```
$ openssl ocsp -index ocsp.index -port 8888 -rsigner ocsp.pem \
>     -rkey ocsp.example.com.key -CA files/ca.crt -text
```

Development documentation:

CHAPTER 9

x509 extensions in other CAs

This page documents the x509 extensions (e.g. for CRLs, etc.) set by other CAs. The information here is used by **django-ca** to initialize and sign certificate authorities and certificates.

Helpful descriptions of the meaning of various extensions can also be found in *x509v3_config(5SSL)* (online).

CommonName

Of course not an extension, but included here for completeness.

CA	Value
Let's Encrypt	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X1
StartSSL	C=IL, O=StartCom Ltd., OU=Secure Digital Certificate Signing, CN=StartCom Certification Authority
StartSSL Class 2	C=IL, O=StartCom Ltd., OU=Secure Digital Certificate Signing, CN=StartCom Class 2 Primary Intermediate Server CA
StartSSL Class 3	C=IL, O=StartCom Ltd., OU=StartCom Certification Authority, CN=StartCom Class 3 OV Server CA
GeoTrust Global	C=US, O=GeoTrust Inc., CN=GeoTrust Global CA
RapidSSL G3	C=US, O=GeoTrust Inc., CN=RapidSSL SHA256 CA - G3
Comodo	C=GB, ST=Greater Manchester, L=Salford, O=COMODO CA Limited, CN=COMODO RSA Certification Authority
Comodo DV	C=GB, ST=Greater Manchester, L=Salford, O=COMODO CA Limited, CN=COMODO RSA Domain Validation Secure Server CA
GlobalSign	C=BE, O=GlobalSign nv-sa, OU=Root CA, CN=GlobalSign Root CA
GlobalSign DV	C=BE, O=GlobalSign nv-sa, CN=GlobalSign Domain Validation CA - SHA256 - G2

authorityInfoAccess

See also:

https://tools.ietf.org/html/rfc5280#section-4.2.2.1

The "CA Issuers" is a URI pointing to the signing certificate. The certificate is in DER/ASN1 format and has a `Content-Type: application/x-x509-ca-cert` header (except where noted).

In CA certificates

Let's Encrypt is notable here because its CA Issuers field points to a pkcs7 file and the HTTP response returns a `Content-Type: application/x-pkcs7-mime` header.

The certificate pointed to by the CA Issuers field is the root certificate (so the Comodo DV CA points to the AddTrust CA that signed the Comodo Root CA).

CA	Value
Let's Encrypt	• OCSP - URI:http://isrg.trustid.ocsp.identrust.com • CA Issuers - URI:http://apps.identrust.com/roots/dstrootcax3.p7c
StartSSL	(not present)
StartSSL Class 2	• OCSP - URI:http://ocsp.startssl.com/ca • CA Issuers - URI:http://aia.startssl.com/certs/ca.crt
StartSSL Class 3	• OCSP - URI:http://ocsp.startssl.com • CA Issuers - URI:http://aia.startssl.com/certs/ca.crt
GeoTrust Global	(not present)
RapidSSL G3	OCSP - URI:http://g.symcd.com
Comodo	OCSP - URI:http://ocsp.usertrust.com
Comodo DV	• CA Issuers - URI:http://crt.comodoca.com/COMODORSAAddTrust • OCSP - URI:http://ocsp.comodoca.com
GlobalSign	(not present)
GlobalSign DV	OCSP - URI:http://ocsp.globalsign.com/rootr1

In signed certificates

Let's Encrypt is again special in that the response has a `Content-Type: application/pkix-cert` header (but at least it's in DER format like every other certificate). RapidSSL uses `Content-Type: text/plain`.

The CA Issuers field sometimes points to the signing certificate (e.g. StartSSL) or to the root CA (e.g. Comodo DV, which points to the AddTrust Root CA)

CA	Value
Let's Encrypt	• OCSP - URI:http://ocsp.int-x1.letsencrypt.org/ • CA Issuers - URI:http://cert.int-x1.letsencrypt.org
StartSSL Class 2	• OCSP - URI:http://ocsp.startssl.com/sub/class2/server/ca • CA Issuers - URI:http://aia.startssl.com/certs/sub.class2.server.ca.crt
StartSSL Class 3	• OCSP - URI:http://ocsp.startssl.com • CA Issuers - URI:http://aia.startssl.com/certs/sca.server3.crt
RapidSSL G3	• OCSP - URI:http://gv.symcd.com • CA Issuers - URI:http://gv.symcb.com/gv.crt
Comodo DV	• CA Issuers - URI:http://crt.comodoca.com/COMODORSADomainV • OCSP - URI:http://ocsp.comodoca.com
GlobalSign DV	• CA Issuers - URI:http://secure.globalsign.com/cacert/gsdomainvalsh • OCSP - URI:http://ocsp2.globalsign.com/gsdomainvalsha2g2

authorityKeyIdentifier

See also:

https://tools.ietf.org/html/rfc5280#section-4.2.1.1

A hash identifying the CA used to sign the certificate. In theory the identifier may also be based on the issuer name and serial number, but in the wild, all certificates reference the *subjectKeyIdentifier*. Self-signed certificates (e.g. Root CAs, like StartSSL and Comodo below) will reference themself, while signed certificates reference the signed CA, e.g.:

Name	subjectKeyIdentifier	authorityKeyIdentifier
Root CA	foo	keyid:foo
Intermediate CA	bar	keyid:foo
Client Cert	bla	keyid:bar

In CA certificates

CA	Value
Let's Encrypt	keyid:C4:A7:B1:A4:7B:2C:71:FA:DB:E1:4B:90:75:FF:C4:15:60:85:89:10
StartSSL	keyid:4E:0B:EF:1A:A4:40:5B:A5:17:69:87:30:CA:34:68:43:D0:41:AE:F2
StartSSL Class 2	keyid:4E:0B:EF:1A:A4:40:5B:A5:17:69:87:30:CA:34:68:43:D0:41:AE:F2
StartSSL Class 3	keyid:4E:0B:EF:1A:A4:40:5B:A5:17:69:87:30:CA:34:68:43:D0:41:AE:F2
GeoTrust Global	keyid:C0:7A:98:68:8D:89:FB:AB:05:64:0C:11:7D:AA:7D:65:B8:CA:CC:4E
RapidSSL G3	keyid:C0:7A:98:68:8D:89:FB:AB:05:64:0C:11:7D:AA:7D:65:B8:CA:CC:4E
Comodo	keyid:AD:BD:98:7A:34:B4:26:F7:FA:C4:26:54:EF:03:BD:E0:24:CB:54:1A
Comodo DV	keyid:BB:AF:7E:02:3D:FA:A6:F1:3C:84:8E:AD:EE:38:98:EC:D9:32:32:D4
GlobalSign	(not present)
GlobalSign DV	keyid:60:7B:66:1A:45:0D:97:CA:89:50:2F:7D:04:CD:34:A8:FF:FC:FD:4B

In signed certificates

CA	Value
Let's Encrypt	keyid:A8:4A:6A:63:04:7D:DD:BA:E6:D1:39:B7:A6:45:65:EF:F3:A8:EC:A1
StartSSL Class 2	keyid:11:DB:23:45:FD:54:CC:6A:71:6F:84:8A:03:D7:BE:F7:01:2F:26:86
StartSSL Class 3	keyid:B1:3F:1C:92:7B:92:B0:5A:25:B3:38:FB:9C:07:A4:26:50:32:E3:51
RapidSSL G3	keyid:C3:9C:F3:FC:D3:46:08:34:BB:CE:46:7F:A0:7C:5B:F3:E2:08:CB:59
Comodo DV	keyid:90:AF:6A:3A:94:5A:0B:D8:90:EA:12:56:73:DF:43:B4:3A:28:DA:E7
GlobalSign DV	keyid:EA:4E:7C:D4:80:2D:E5:15:81:86:26:8C:82:6D:C0:98:A4:CF:97:0F

basicConstraints

See also:

https://tools.ietf.org/html/rfc5280#section-4.2.1.9

The `basicConstraints` extension specifies if the certificate can be used as a certificate authority. It is always marked as critical. The `pathlen` attribute specifies the levels of possible intermediate CAs. If not present, the level of intermediate CAs is unlimited, a `pathlen:0` means that the CA itself can not issue certificates with `CA:TRUE` itself.

In CA certificates

CA	Value
Let's Encrypt	(critical) CA:TRUE, pathlen:0
StartSSL	(critical) CA:TRUE
StartSSL Class 2	(critical) CA:TRUE, pathlen:0
StartSSL Class 3	(critical) CA:TRUE, pathlen:0
GeoTrust Global	(critical) CA:TRUE
RapidSSL G3	(critical) CA:TRUE, pathlen:0
Comodo	(critical) CA:TRUE
Comodo DV	(critical) CA:TRUE, pathlen:0
GlobalSign	(critical) CA:TRUE
GlobalSign DV	(critical) CA:TRUE, pathlen:0

In signed certificates

CA	Value
Let's Encrypt	(critical) CA:FALSE
StartSSL Class 2	(critical) CA:FALSE
StartSSL Class 3	CA:FALSE
RapidSSL G3	(critical) CA:FALSE
Comodo DV	(critical) CA:FALSE
GlobalSign DV	CA:FALSE

crlDistributionPoints

See also:

https://tools.ietf.org/html/rfc5280#section-4.2.1.13

In theory a complex multi-valued extension, this extension usually just holds a URI pointing to a Certificate Revokation List (CRL).

Root certificate authorities (StartSSL, GeoTrust Global, GlobalSign) do not set this field. This usually isn't a problem since clients have a list of trusted root certificates anyway, and browsers and distributions should get regular updates on the list of trusted certificates.

All CRLs linked here are all in DER/ASN1 format, and the `Content-Type` header in the response is set to `application/pkix-crl`. Only Comodo uses `application/x-pkcs7-crl`, but it is also in DER/ASN1 format.

In CA certificates

CA	Value	Content-Type
Let's Encrypt	URI:http://crl.identrust.com/DSTROOTCAX3CRL.crl	application/pkix-crl
StartSSL	(not present)	
StartSSL Class 2	URI:http://crl.startssl.com/sfsca.crl	application/pkix-crl
StartSSL Class 3	URI:http://crl.startssl.com/sfsca.crl	application/pkix-crl
GeoTrust Global	(not present)	
RapidSSL G3	URI:http://g.symcb.com/crls/gtglobal.crl	application/pkix-crl
Comodo	URI:http://crl.usertrust.com/AddTrustExternalCARoot.crl	application/x-pkcs7-crl
Comodo DV	URI:http://crl.comodoca.com/COMODORSACertificationAuthority.crl	application/x-pkcs7-crl
GlobalSign	(not present)	
GlobalSign DV	URI:http://crl.globalsign.net/root.crl	application/pkix-crl

In signed certificates

Let's Encrypt is so far the only CA that does not maintain a CRL for signed certificates. Major CAs usually don't fancy CRLs much because they are a large file (e.g. Comodos CRL is 1.5MB) containing all certificates and cause major traffic for CAs. OCSP is just better in every way.

CA	Value	Content-Type
Let's Encrypt	(not present)	
StartSSL Class 2	URI:http://crl.startssl.com/crt2-crl.crl	application/pkix-crl
StartSSL Class 3	URI:http://crl.startssl.com/sca-server3.crl	application/pkix-crl
RapidSSL G3	URI:http://gv.symcb.com/gv.crl	application/pkix-crl
Comodo DV	URI:http://crl.comodoca.com/COMODORSADomainValidationSecureServerCA.crl	application/x-pkcs7-crl
GlobalSign DV	URI:http://crl.globalsign.com/gs/gsdomainvalsha2g2.crl	application/pkix-crl

extendedKeyUsage

A list of purposes for which the certificate can be used for. CA certificates usually do not set this field.

In CA certificates

CA	Value
Let's Encrypt	(not present)
StartSSL	(not present)
StartSSL Class 2	(not present)
StartSSL Class 3	TLS Web Client Authentication, TLS Web Server Authentication
GeoTrust Global	(not present)
RapidSSL G3	(not present)
Comodo	(not present)
Comodo DV	TLS Web Server Authentication, TLS Web Client Authentication
GlobalSign	(not present)
GlobalSign DV	(not present)

In signed certificates

CA	Value
Let's Encrypt	TLS Web Server Authentication, TLS Web Client Authentication
StartSSL Class 2	TLS Web Client Authentication, TLS Web Server Authentication
StartSSL Class 3	TLS Web Client Authentication, TLS Web Server Authentication
RapidSSL G3	TLS Web Server Authentication, TLS Web Client Authentication
Comodo DV	TLS Web Server Authentication, TLS Web Client Authentication
GlobalSign DV	TLS Web Server Authentication, TLS Web Client Authentication

issuerAltName

See also:

https://tools.ietf.org/html/rfc5280#section-4.2.1.7

Only StartSSL sets this field in its signed certificates. It's a URI pointing to their homepage.

In CA certificates

CA	Value
Let's Encrypt	(not present)
StartSSL	(not present)
StartSSL Class 2	(not present)
StartSSL Class 3	(not present)
GeoTrust Global	(not present)
RapidSSL G3	(not present)
Comodo	(not present)
Comodo DV	(not present)
GlobalSign	(not present)
GlobalSign DV	(not present)

In signed certificates

CA	Value
Let's Encrypt	(not present)
StartSSL Class 2	URI:http://www.startssl.com/
StartSSL Class 3	URI:http://www.startssl.com/
RapidSSL G3	(not present)
Comodo DV	(not present)
GlobalSign DV	(not present)

keyUsage

See also:

https://tools.ietf.org/html/rfc5280#section-4.2.1.3

List of permitted key usages. Usually marked as critical, except for certificates signed by StartSSL.

In CA certificates

CA	Value
Let's Encrypt	(critical) Digital Signature, Certificate Sign, CRL Sign
StartSSL	(critical) Certificate Sign, CRL Sign
StartSSL Class 2	(critical) Certificate Sign, CRL Sign
StartSSL Class 3	(critical) Certificate Sign, CRL Sign
GeoTrust Global	(critical) Certificate Sign, CRL Sign
RapidSSL G3	(critical) Certificate Sign, CRL Sign
Comodo	(critical) Digital Signature, Certificate Sign, CRL Sign
Comodo DV	(critical) Digital Signature, Certificate Sign, CRL Sign
GlobalSign	(critical) Certificate Sign, CRL Sign
GlobalSign DV	(critical) Certificate Sign, CRL Sign

In signed certificates

CA	Value
Let's Encrypt	(critical) Digital Signature, Key Encipherment
StartSSL Class 2	Digital Signature, Key Encipherment, Key Agreement
StartSSL Class 3	Digital Signature, Key Encipherment
RapidSSL G3	(critical) Digital Signature, Key Encipherment
Comodo DV	(critical) Digital Signature, Key Encipherment
GlobalSign DV	(critical) Digital Signature, Key Encipherment

subjectAltName

The `subjectAltName` extension is not present in any CA certificate, and of course whatever the customer requests in signed certificates.

In CA certificates

CA	Value
Let's Encrypt	•
StartSSL	•
StartSSL Class 2	•
StartSSL Class 3	•
GeoTrust Global	•
RapidSSL G3	•
Comodo	•
Comodo DV	•
GlobalSign	•
GlobalSign DV	•

subjectKeyIdentifier

See also:

https://tools.ietf.org/html/rfc5280#section-4.2.1.2

The subjectKeyIdentifier extension provides a means of identifying certificates. It is a mandatory extension for CA certificates. Currently only RapidSSL does not set this for signed certificates.

The value of the subjectKeyIdentifier extension reappears in the *authorityKeyIdentifier* extension (prefixed with `keyid:`).

In CA certificates

CA	Value
Let's Encrypt	A8:4A:6A:63:04:7D:DD:BA:E6:D1:39:B7:A6:45:65:EF:F3:A8:EC:A1
StartSSL	4E:0B:EF:1A:A4:40:5B:A5:17:69:87:30:CA:34:68:43:D0:41:AE:F2
StartSSL Class 2	11:DB:23:45:FD:54:CC:6A:71:6F:84:8A:03:D7:BE:F7:01:2F:26:86
StartSSL Class 3	B1:3F:1C:92:7B:92:B0:5A:25:B3:38:FB:9C:07:A4:26:50:32:E3:51
GeoTrust Global	C0:7A:98:68:8D:89:FB:AB:05:64:0C:11:7D:AA:7D:65:B8:CA:CC:4E
RapidSSL G3	C3:9C:F3:FC:D3:46:08:34:BB:CE:46:7F:A0:7C:5B:F3:E2:08:CB:59
Comodo	BB:AF:7E:02:3D:FA:A6:F1:3C:84:8E:AD:EE:38:98:EC:D9:32:32:D4
Comodo DV	90:AF:6A:3A:94:5A:0B:D8:90:EA:12:56:73:DF:43:B4:3A:28:DA:E7
GlobalSign	60:7B:66:1A:45:0D:97:CA:89:50:2F:7D:04:CD:34:A8:FF:FC:FD:4B
GlobalSign DV	EA:4E:7C:D4:80:2D:E5:15:81:86:26:8C:82:6D:C0:98:A4:CF:97:0F

In signed certificates

CA	Value
Let's Encrypt	F4:F3:B8:F5:43:90:2E:A2:7F:DD:51:4A:5F:3E:AC:FB:F1:33:EE:95
StartSSL Class 2	C7:AA:D9:A4:F0:BC:D1:C1:1B:05:D2:19:71:0A:86:F8:58:0F:F0:99
StartSSL Class 3	F0:72:65:5E:21:AA:16:76:2C:6F:D0:63:53:0C:68:D5:89:50:2A:73
RapidSSL G3	(not present)
Comodo DV	F2:CB:1F:E9:6E:D5:43:E3:85:75:98:5F:97:7C:B0:59:7F:D5:C0:C0
GlobalSign DV	52:5A:45:5B:D4:9D:AC:65:30:BD:67:80:6C:D1:A1:3E:09:F7:FD:92

Other extensions

Extensions used by certificates encountered in the wild that django-ca does not (yet) support in any way.

In CA certificates

CA	Value
Let's Encrypt	X509v3 Certificate Policies, X509v3 Name Constraints
StartSSL	X509v3 Certificate Policies, Netscape Cert Type, Netscape Comment
StartSSL Class 2	X509v3 Certificate Policies
StartSSL Class 3	X509v3 Certificate Policies
GeoTrust Global	(none)
RapidSSL G3	X509v3 Certificate Policies
Comodo	X509v3 Certificate Policies
Comodo DV	X509v3 Certificate Policies
GlobalSign	(none)
GlobalSign DV	X509v3 Certificate Policies

In signed certificates

CA	Value
Let's Encrypt	X509v3 Certificate Policies
StartSSL Class 2	X509v3 Certificate Policies
StartSSL Class 3	X509v3 Certificate Policies
RapidSSL G3	X509v3 Certificate Policies
Comodo DV	X509v3 Certificate Policies
GlobalSign DV	X509v3 Certificate Policies

CHAPTER 10

ChangeLog

1.4.1 (to be released)

- Update requirements.
- Use Travis CI for continuous integration. **django-ca** is now tested with Python 2.7, 3.4, 3.5, 3.6 and nightly, using Django 1.8, 1.9 and 1.10.
- Fix a few test errors for Django 1.8.
- Examples now consistently use 4096 bit certificates.
- Some functionality is now migrated to `cryptography` in the ongoing process to deprecate pyOpenSSL (which is no longer maintained).
- OCSPView now supports directly passing the public key as bytes. As a consequence, a bad certificate is now only detected at runtime.

1.4.0 (2016-09-09)

- Make sure that Child CAs never expire after their parents. If the user specifies an expiry after that of the parent, it is silently changed to the parents expiry.
- Make sure that certificates never expire after their CAs. If the user specifies an expiry after that of the parent, throw an error.
- Rename the `--days` parameter of the `sign_cert` command to `--expires` to match what we use for `init_ca`.
- Improve help-output of `--init-ca` and `--sign-cert` by further grouping arguments into argument groups.
- Add ability to add CRL-, OCSP- and Issuer-URLs when creating CAs using the `--ca-*` options.
- Add support for the `nameConstraints` X509 extension when creating CAs. The option to the `init_ca` command is `--name-constraint` and can be given multiple times to indicate multiple constraints.
- Add support for the `tlsfeature` extension, a.k.a. "TLS Must Staple". Since OpenSSL 1.1 is required for this extension, support is currently totally untested.

1.3.0 (2016-07-09)

- Add links for downloading the certificate in PEM/ASN format in the admin interface.

- Add an extra chapter in documentation on how to create intermediate CAs.
- Correctly set the issuer field when generating intermediate CAs.
- `fab init_demo` now actually creates an intermediate CA.
- Fix help text for the `--parent` parameter for `manage.py init_ca`.

1.2.2 (2016-06-30)

- Rebuild to remove old migrations accidentally present in previous release.

1.2.1 (2016-06-06)

- Add the `CA_NOTIFICATION_DAYS` setting so that watchers don't receive too many emails.
- Fix changing a certificate in the admin interface (only watchers can be changed at present).

1.2.0 (2016-06-05)

- **django-ca** now provides a complete OCSP responder.
- Various tests are now run with a pre-computed CA, making tests much fater and output more predictable.
- Update lots of documentation.

1.1.1 (2016-06-05)

- Fix the `fab init_demo` command.
- Fix installation via `setup.py install`, fixes #2 and #4. Thanks to Jon McKenzie for the fixes!

1.1.0 (2016-05-08)

- The subject given in the `manage.py init_ca` and `manage.py sign_cert` is now given in the same form that is frequently used by OpenSSL, "/C=AT/L=...".
- On the command line, both CAs and certificates can now be named either by their CommonName or with their serial. The serial can be given with only the first few letters as long as it's unique, as it is matched as long as the serial starts with the given serial.
- Expiry time of CRLs can now be specified in seconds. `manage.py dump_crl` now uses the `--expires` instead of the old `--days` parameter.
- The admin interface now accounts for cases where some or all CAs are not useable because the private key is not accessable. Such a scenario might occur if the private keys are hosted on a different machine.
- The app now provides a generic view to generate CRLs. See *Use generic view to host a CRL* for more information.
- Fix the display of the default value of the –ca args.

- Move this ChangeLog from a top-level .md file to this location.
- Fix shell example when issueing certificates.

1.0.1 (2016-04-27)

- Officially support Python2.7 again.
- Make sure that certificate authorities cannot be removed via the web interface.

1.0.0 (2016-04-27)

This represents a massive new release (hence the big version jump). The project now has a new name (**django-ca** instead of just "certificate authority") and is now installable via pip. Since versions prior to this release probably had no users (as it wasn't advertised anywhere), it includes several incompatible changes.

General

- This project now runs under the name **django-ca** instead of just "certificate authority".
- Move the git repository is now hosted at https://github.com/mathiasertl/django-ca.
- This version now absolutely assumes Python3. Python2 is no longer supported.
- Require Django 1.8 or later.
- django-ca is now usable as a stand-alone project (via git) or as a reusable app (via pip).

Functionality

- The main app was renamed from `certificate` to `django_ca`. See below for how to upgrade.

manage.py interface

- `manage.py` commands are now renamed to be more specific:
  - `init` -> `init_ca`
  - `sign` -> `sign_cert`
  - `list` -> `list_certs`
  - `revoke` -> `revoke_cert`
  - `crl` -> `dump_crl`
  - `view` -> `view_cert`
  - `watch` -> `notify_expiring_certs`
  - `watchers` -> `cert_watchers`
- Several new `manage.py` commands:
  - `dump_ca` to dump CA certificates.

  - `dump_cert` to dump certificates to a file.
  - `dump_ocsp_index` for an OCSP responder, `dump_crl` no longer outputs this file.
  - `edit_ca` to edit CA properties from the command line.
  - `list_cas` to list available CAs.
  - `view_ca` to view a CA.
- Removed the `manage.py remove` command.
- `dump_{ca,cert,crl}` can now output DER/ASN1 data to stdout.

0.2.1 (2015-05-24)

- Signed certificates are valid five minutes in the past to account for possible clock skew.
- Shell-scripts: Correctly pass quoted parameters to manage.py.
- Add documentation on how to test CRLs.
- Improve support for OCSP.

0.2 (2015-02-08)

- The `watchers` command now takes a serial, like any other command.
- Reworked `view` command for more robustness.
  - Improve output of certificate extensions.
  - Add the `-n`/`--no-pem` option.
  - Add the `-e`/`--extensions` option to print all certificate extensions.
  - Make output clearer.
- The `sign` command now has
  - a `--key-usage` option to override the `keyUsage` extended attribute.
  - a `--ext-key-usage` option to override the `extendedKeyUsage` extended attribute.
  - a `--ocsp` option to sign a certificate for an OCSP server.
- The default `extendedKeyUsage` is now `serverAuth`, not `clientAuth`.
- Update the remove command to take a serial.
- Ensure restrictive file permissions when creating a CA.
- Add requirements-dev.txt

0.1 (2015-02-07)

- Initial release

CHAPTER 11

Development

Setup demo

You can set up a demo using `fab init_demo`. First create a minimal `localsettings.py` file (in `ca/ca/localsettings.py`):

```
DEBUG = True
SECRET_KEY = "whatever"
```

And then simply run `fab init_demo` from the root directory of your project.

Run test-suite

To run the test-suite, simply execute:

```
python setup.py test
```

... or just run some of the tests:

```
python setup.py test --suite=tests_command_dump_crl
```

To generate a coverate report:

```
python setup.py coverage
```

Useful OpenSSL commands

CRLs

Convert a CRL to text on stdout:

```
openssl crl -inform der -in sfsca.crl -noout -text
```

Convert a CRL to PEM to a file:

```
openssl crl -inform der -in sfsca.crl -outform pem -out test.pem
```

Verify a certificate using a CRL:

```
openssl verify -CAfile files/ca_crl.pem -crl_check cert.pem
```

OCSP

Run a OCSP responder:

```
openssl ocsp -index files/ocsp_index.txt -port 8888 \
   -rsigner files/localhost.pem -rkey files/localhost.key \
   -CA ca.pem -text
```

Verify a certificate using OCSP:

```
openssl ocsp -CAfile ca.pem -issuer ca.pem -cert cert.pem \
    -url http://localhost:8888 -resp_text
```

Other

Convert a p7c/pkcs7 file to PEM (Let's Encrypt CA Issuer field) (see also *pkcs7(1SSL)* - online):

```
openssl pkcs7 -inform der -in letsencrypt.p7c -print_certs \
   -outform pem -out letsencrypt.pem
```

Development webserver via SSL

To test a certificate in your webserver, first install the root certificate authority in your browser, then run `stunnel4` and `manage.py runserver` in two separate shells:

```
stunnel4
HTTPS=1 python manage.py runserver 8001
```

Then visit https://localhost:8443.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

C

D

E

O

R

T