django-ca Documentation

Release 1.0.1

Mathias Ertl

July 09, 2016

Contents

django-ca is a small project to manage TLS certificate authorities and easily issue 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:

- Set up a secure local certificate authority in just a few minutes.
- Written in Python2.7/Python3.4+.
- Manage your entire certificate authority from the command line and/or via Djangos admin interface.
- Get email notifications about certificates about to expire.
- Support generating for certificate revocation lists (CRLs).
- Generates index files that can be used with the *openssl ocsp* command for a crude OCSP service.

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*.

1.1 Requirements

- Python 2.7 or Python 3.4+
- Django 1.8+
- Any database supported by Django (sqlite3/MySQL/PostgreSQL/...)

1.2 As Django app (in an your 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 "Root CA" AT Vienna Vienna Org OrgUnit ca.example.com
```

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

1.3 As standalone project

In this variant, you can run **django-ca** stand-alone. You can use the project strictly from the command line, the webinterface is completely optional.

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
```

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. After you have configured **django-ca** (especially `SECRET_KEY`, `DATABASES` and, if you intend to use the webinterface, `STATIC_ROOT`), you need to run 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 AT Vienna Vienna Org OrgUnit 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.

1.3.1 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} $@
```

1.3.2 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.

1.4 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

2.1 Update from pre 1.0.0

If you're updating from a version earlier then 1.0.0 (which was the first real release), you have to reset the 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

CHAPTER 3

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 via manage.py commands. 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 \
   "Test CA" AT Vienna Vienna Example ExampleUnit ca.example.com
$ python manage.py list_cas
BD:5B:AB:5B:A2:1C:49:0D:9A:B2:AA:BC:68:ED:ED:7D - Test CA
$ python manage.py view_ca BD:5B:AB:5B:A2:1C:49:0D:9A:B2:AA:BC:68:ED:ED:7D \
   | grep OCSP
* OCSP URL: http://ocsp.ca.example.com
$ python manage.py edit_ca --ocsp-url=http://new-ocsp.ca.example.com
$ python manage.py view_ca BD:5B:AB:5B:A2:1C:49:0D:9A:B2:AA:BC:68:ED:ED:7D \
   | grep OCSP
* OCSP URL: http://new-ocsp.ca.example.com
```

CHAPTER 4

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_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,
}
```

CHAPTER 5

Custom manage.py commands

You can run your entire CA from the console without any webinterface at all, using Djangos `manage.py` script. In case you also run a webinterface, this will of course act with the same settings, database and CA certificates.

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 use **django-ca** *as a standalone project*.

Execute `manage.py <subcommand> -h` to get help on the subcommand. Here is an overview of all subcommands provided by **django-ca**:

Command	Description
cert_watchers	Add/remove addresses to be notified of an expiring certificate.
dump_cert	Dump a certificate to a file.
dump_crl	Write the certificate revocation list (CRL).
dump_ocsp_index	Write an OCSP index 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.

The following commands are used to manage certificate autorities:

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

CHAPTER 6

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.

6.1 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
```

6.2 Generate the CRL

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 `--days=N`), so you must periodically regenerate it, e.g. via a cron-job.

6.3 Host the CRL

How and where to host that 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 7

Run a OCSP responder

Hosting an OCSP service is the second method (besides `CRLs <crl`) of letting a client know if a certificate has been revoked.

django-ca does not provide a way to host an OCSP service itself, but all other necessary parts are included, if you intend to run such a service.

7.1 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.example.com.key 4096
openssl req -new -key ocsp.example.com.key -out ocsp.example.com.csr -utf8 \
   -batch -subj '/CN=ocsp.example.com'
python manage.py sign_cert --csr=ocsp.example.com.csr \
   --out=ocsp.example.com.pem --cn=ocsp.example.com --ocsp
```

7.2 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
```

7.3 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.example.com.pem \
   -rkey ocsp.example.com.key -CA files/ca.crt -text
```

CHAPTER 8

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).

8.1 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).

8.1.1 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

8.1.2 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

8.2 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

8.2.1 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

8.2.2 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

8.3 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.

8.3.1 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

8.3.2 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

8.4 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.

8.4.1 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

8.4.2 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

8.5 extendedKeyUsage

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

8.5.1 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)

8.5.2 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

8.6 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.

8.6.1 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)

8.6.2 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)

8.7 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.

8.7.1 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

8.7.2 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

8.8 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:`).

8.8.1 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

8.8.2 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

8.9 Other extensions

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

8.9.1 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

8.9.2 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 9

Development

9.1 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.

9.2 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
```

9.3 Useful OpenSSL commands

9.3.1 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
```

9.3.2 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
```

9.3.3 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
```

9.4 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 10

Indices and tables

- genindex
- modindex
- search