ssl-mgr

Release 6.2.0

Gene C

Jul 10, 2025

CONTENTS:

CHAPTER

ONE

SSL-MGR

1.1 Overview

Certificate management tool.

By way of background, I wrote this with 3 goals. Specifically to:

- Simplify certificate management - (i.e. automatic, simple and robust)
- Support *dns-01* acme challenge with Letsencrypt (as well as *http-01*)
- Support *DANE TLS*

The aim is to make things as robust, complete and as simple to use as possible. Under the hood, make it sensible and automate wherever it is feasible. A good tool does things correctly and makes it straightforward and as simple as possible; but no simpler.

In practical terms, there are only 2 common commands that are needed with *sslm-mgr*:

- **renew** - creates new certificate(s) in *next* : current certs remain in *curr*.
- **roll** - moves *next* to become the new *curr*.

Once things are set up these can be run out of cron - renew, then wait, then roll. Clean and simple. Strictly speaking, rolling of certs is only needed when they are advertized via DNS (say). Rolling provides for both old and new keys to be made available for some period while DNS servers update. The last step is to advertize new certs only. Changing to new certs without rolling can be problematic if some DNS servers still point to the older certs.

While there are other command line options, the **-s, –status** option provides a convenient view of all managed certificates along with their expiration and time remaining before renewal. The **sslm-info** standalone program provides a convenient way to display information about certs (or chains of certs), CSRs etc.

N.B. DNSSEC is required for DANE otherwise it is not needed. However, I do recommend using DNSSEC and I have made available the tool I use to manage it DNS/DNSSEC[4].

DANE can use either self-signed certs or known CA signed certs. *ssl-mgr* makes it straightforward to create self-signed certs as well. However, in practice, it is safer to use CA signed certs for SMTP to avoid delivery problems in the event some servers require CA chain of trust. We therefore recommend using CA signed certificates and therefore publishing DANE TLSA records using those certificiates. Each MX will have its own TLSA record.

While DANE may be used for other TLS services, such as https, in practice it is only used with email.

For convenience, there is a PDF version of this document in the Docs directory.

Note:

[4] dns_tools : https://github.com/gene-git/dns_tools

All git tags are signed with arch@sapience.com key which is available via WKD or download from https://www.sapience.com/tech. Add the key to your package builder gpg keyring. The key is included in the Arch package and the source= line with *?signed* at the end can be used to verify the git tag. You can also manually verify the signature

1.2 Key Features

- Handles creating new and renewing certificates
- Generates key pairs and Certificate Signing Request, CSR, to provide maximum control
- Supports http-01 and dns-01 acme challenges
- Outputs DNS files for acme DNS-01 authentication as well as optioanal DANE TLSA files. These files are to be included by the apex domain zone file. This makes updates straightforward.
- Uses certbot in manual mode to handle communication with letsencrypt, account tracking etc.
- Processes multiple domains, where each domain each can have multiple certs for different purposes.

1.3 New / Interesting

Recent changes important info.

Version 6.1 :

- New integrity check.

  On each run *sslm-mgr* validates that the production directory is up to date and consistent with the current suite of certificates, keys and TLSA files.

  If not, it explains what the problem is and suggests possible ways to proceed.

  Note that the first run after updating to *6.1* it will automatically re-sync production directory if necessary. No action is required by you.

  For more details see *Dealing with Possible Problems*.

- Keep certs and production certs fully synced.

  Includes removing *next* directory from production after the *roll* has happened and *next* is no longer needed. This change allows us to check that production is correctly synchronized. Earlier versions did not remove any files from production, needed or not.

- New dev option *–force-server-restarts*.
- Add ability to specify the top level directory (where configs and outputs are read from / saved to) via environment variable *SSL_MGR_TOPDIR*.
- External programs are run using a local copy of *run_prog()* from the *pyconcurrent* module. You can also install *pyconcurrent* which will ensure the latest version is always used. It is available in Github pyconcurrnet and AUR pyconcurrnet.

Version 6.0 : Major Changes

- PEP-8, PEP-257 and PEP-484 style and type annotations.
- Major re-write and tidy ups.
- Split up various modules (e.g. certs -> 5 separate crypto modules.)
- Ensure config and command line options are 100% backward compatible.

- Improve 2 config values:

  Background: Local CAs have self-signed a root CA certificate which is then used to sign an intermediate CA cert. The intermediate CA is in turn used to sign application certificates.

  - ca-info.conf: Intermediate local CA entries.

    ca_type = "local" is preferred to "self" (NB both work). "self" should still be used for self-signed root CAs where it makes more sense. Intermediate are signed by root and are therefore not self-signed.

  - CA service config file for self-signed root certificate:

    "signing_ca" = "self" is now preferred to an empty string (NB Both work).

  - These 2 changes are optional but preferred. No other config file changes.

- Simplify logging code.

Older Changes:

- Support Letsencrypt alternate root chain.

  Set via *ca_preferred_chain* option in *ca-info.conf* file (see example file).

  By default LE root cert is *ISRG Root X1* (RSA). Since it is standard to use ECC for certificates, it is preferable to use LE *ISRG Root X2* (ECC) which is smaller and faster since less data is exchanged during TLS handshake.

  X2 cert is cross-signed by X1 cert, so any client trusting X1 should trust X2.

  Some more info here: LE Certificates: and Compatibility.

- New config option *post_copy_cmd*

  For each server getting copies of certs may run this command on machine on which sslm-mgr is running. The command is passed server hostname as an argument. Usage Example: if a server needs a file permission change for an application user to read private key(s). This option is a list of *[server-host, command]* pairs. See *Config ssl-mgr.conf*

- X509v3 Extended Key Usage adds "Time Stamping"

- Changed sslm-dhparm to generate RFC-7919 Negotiated Finite Field Diffie-Hellman Ephemeral Parameters files - with the default now set to ffdhe8192 instead of ffdhe4096. User options -k overrides the default as usual

  NB If you manually update DH files in prod-certs, then push to all servers:

  ```
  sslm-mgr dev -certs-prod
  ```

  NB TLSv1.3 restricts DH key exchange to named groups only.

- openssl trusted certificates there is ExtraData after the cert which has the trust data. cryptography.x509 will not load this so strip it off. see : https://github.com/pyca/cryptography/issues/5242

- Add a working example of self signed web cert in examples/ca-self. Create ca-certs (./make-ca) then generate new web cert signed by that ca. (sslm-mgr -renew; sslm-mgr -roll)

- letsencrypt dns-01 challenge may not always use the apex domain's authoritative servers or perhaps their (secondary) checks might lag more. We tackle this with the addition of 2 new variables to the top level config:

  - *dns-check-delay*.

    Given in seconds, this causes a delay before attempting to validate that all authoritative servers have up to date acme challenge dns txt records. Defaults to 240 seconds - this may well need to be made longer. Obviously, this does lead to longer run times - by design.

  - *dns_xtra_ns*.

List of nameservers (hostname or ip) which will be checked to have up to date acme challenge dns txt records in addition to each apex domain authoritative nameserver. Default value is:

dns_xtra_ns = ['1.1.1.1', '8.8.8.8', '9.9.9.9', '208.67.222.222']

- improve the way nameservers are checked for being up to date with acme challenges. First check the primary has all the acme challenge TXT records. Then check all nameservers, including the *xtra_ns* have the same serial as the primary
- While things can take longer than previous versions, teting to date has shown it to be robust and working well with letsencrypt.

1.4 More Detail

The tool keeps and manages 2 versions of every set of data. Each set of data is comprised of certificates, keys, CSRs, etc.

One version of the data has the current (aka *curr*) set and the other has the next set (aka *next*). *curr* are those currently in use while *next* are those that are on deck to become the next current set.

Key rolling is standard practice and should be familiar to those who have implemented *DNSSEC*. A *roll* is a robust method of updating keys/certs with new ones in a way that ensures nothing breaks.

The current key/cert is always advertised in DNS. After creating new keys/certs, DNS is then upated to advertise both the current and the newly created next ones.

An appropriate amount of time needs to pass with both current and next in DNS before doing the *roll*. This gives the time needed for DNS servers to refresh. Once refreshed, the DNS servers now have both the current and the next set of keys/certs.

After sufficient time, update a second time, and now only the new keys (the new current ones) are advertised in DNS.

A *roll* is required for *DNSSEC* as well as for *DANE*, which we manage.

Without any loss of functionality and to keep things nice and simple, we treat every update as requiring a key roll.

Again, a *roll* is required for *DANE TLS* but is not needed for things such as web server certificate update.

If you are not advertizing certificate info using DNS servers (e.g. DNSSEC, DANE) then there is no need have any delay between making a new certificate using *-renew* and doing the *-roll*.

In this case, you can set the config variable *min_roll_mins* to **0** minutes. The default min roll time is 90 minutes. And if automating (via cron or similar) then you can also use a smaller do the *roll* immediately after the *renew* as well. In cron you could have roll set to run 1 minutes after the renew.

Furthermore, you are always in control and, should it be needed, you can do whatever you choose.

e.g. Using *-f* will force things to happen (a roll or create new certs and so on.)

1.4.1 Curr & Next

These are kept in directories that contain different versions of the same set of files. Of course *next* has newer versions. For example for the group *example.com* and the service *web-ec* these directories would be located in:

```
certs/example.com/web-rc/curr/
certs/example.com/web-rc/next/
```

In order of creation these are:

File	What
privkey.pem	the private key
csr.pem	certificate signing request
cert.pem	certificate
chain.pem	CA root + intermediate certs
fullchain.pem	Our cert.pem + CA chain
bundle.pem	Our privkey + fullchain
info	Contains date/time when next was rolled to curr (curr only)

Once config is setup, a cron/timer to run *renew* followed by *roll* 2 or 3 hours later should take care of everything. Can be run daily or weekly.

The *curr/next* directories will also be copied to the production directory, as specified in *conf.d/ssl-mgr.conf* by the variable *prod_cert_dir*.

1.4.2 Diffie-Hellman Parameters

There is also a tool, *sslm-dhparm*, which generates Diffie-Hellman parameters. This can be added to the cron file.

By default *sslm-dhparm* only generates new parameters if they are more than 120 days old, or absent. This can therefore be run weekly without issues.

Note: The new, preferred and now default DH parameters are based on RFC-7919 rfc_7919 pre-defined named groups. The default is *ffdhe4096*. Pre-defined named groups only need to be generated once and will only be generated if absent.

Strictly these don't need to be in cron, but its convenient to have the program check and create DH parameters should the file be missing. May happen occasionally after adding new domain.

The 6 month default refresh, ony applies for non RFC-7919 params, and is recommended because it can be a bit time consuming to generate them. Actual time varies with key size.

When using a pre-defined named group (e.g. *ffdhe4096*), it is very quick to produce and tool simply checks if file exists without any age requirement. These are only created once.

Sample cron files are provided in the examples directory.

1.4.3 More Details

There are several additional commands that offer fine grained control, in case its needed. These are discussed in detail below. One example is the *-f* or *–force* option which does what the name suggests.

The tool handles keys, certificate signing requests (CSR) and certs. It also takes care of generating DANE TLSA DNS records should you want to use them and reloads/restarts specific servers whenever they need it. Each server has defined dependencies which trigger restarts whenever those dependencies have changed.

For example, a web server may depend on one or more apex domain certificates and will be restarted when any of those certs change.

It needs external support tools such as zone signing for DNSSEC and restarting dns servers as well as reloading web or mail servers to ensure new certs are picked up. These are provided via the top level config file.

There is support for private/self-signed CAs and Letsencrypt CA. Letsencrypt acme validation challenges[3] can use either http or dns; dns is preferred whenever possible.

[3] acme-challenge : https://letsencrypt.org/docs/challenge-types/

1.4.4 DANE

For DANE TLSA records, care must be taken to properly *roll* new keys. Key rolling ensures that the *next* key and the *curr* key are both advertised in DNS for some period. After some time the new key can be made *curr*. This waiting period should be long enough to provide sufficient time for all DNS servers to pick up both old and new new keys before DNS is changed to only show the new ones. It's reasonable to wait 2 x the DNS TTL or longer.

After that wait time, the new (*next*) keys can be then be made available as the new *curr* ones. Applications, mail really, can now use the new keys since the world has both sets of keys.

Then DNS servers can then be updated again, this time with just the new (now *curr*) keys in the TLSA records. DANE key roll is similar to key roll for DNSEC. DANE TLSA actually requires DNSSEC.

DANE was designed as an alternative to third party certificate authorities like letsencrypt which means its fine to used self signed or CA signed certs. While DANE could be used for web servers to date it is really only used for email.

The companion *dns_tools* package takes care of all our DNSSEC needs[Page 1, 4]:

And I recommend using it to simplify the DNS refresh needed for validating Letsencrypt acme challenges using *DNS-01* as well as for DANE TLSA. A DNS refresh means resign zones (when using DNSSEC) and then restarting the primary dns server.

DANE TLSA records contain the public key, or a hash of that key, and thus need to be refreshed whenever that key changes; this is the key roll. It also means that if the key is kept the same, then the TLSA records aren't changing[5]. *ssl-mgr* has an option to re-use the public key when certs are being renewed, and this allows the TLSA records to remain unchanged. In that case no key roll is needed until that key is changed. Some may find this useful.

It basically means using the same certificate signing request, CSR, to get a new cert. The CSR contains the public key associated with the private key. So if keys dont change CSR doesn't change either, and the same CSR can be re-used.

However, I find *ssl-mgr* makes it so simple to renew with new keys, that I don't see much point in reusing the old keys. Of course using new keys offers a security benefit.

Note that each MX for the mail domain will have a TLSA record as required by the standard.

1.4.5 Acme Challenge

Using *DNS-01* to validate Letsencrypt acme challenges is done by adding the challenge TXT records to DNS, signing the zones (if using DNSSEC) and pushing them out, so that Letsencrypt can subsequently check those DNS records match appropriately and then they provide the requested cert. Some tool to do that DNS refresh is needed for this pupose. I use dns_tools to do that. DNS refresh also happens after DANE TLSA records are updated.

This should run on the DNS signing server. This allows files with DNS records, acme challenges and TLSA, to be written to accessible directories on same machine. I may enhance this to allow the dns signing server to be remote, some day.

[5] DANE can use either public key or the cert. Cert does change when it's reneweed even if the public key is unchanged. I believe pretty much everyone uses the public key not the cert in TLSA reords.

CHAPTER

TWO

GETTING STARTED

The first order of business is creating the config files. These specify everything that's needed.

This includes the shared config *ssl-mgr.conf* which includes the commands to restart servers (web, mail), where to put the acme challenge files (web or dns) and where the final certificates are to be stored.

Each certificate to be issued has it's own *service* config file.

The sample configs provided in examples/conf.d provide a template to get started.

2.1 Tools

The main tool for generating and managing certificates is *sslm-mgr*. As usual, help is available using *-h*.

There is also a dev mode, providing access to some lower lever tasks. You probably should seldom, if ever, need dev mode, but in case you do, it is activated by using the *dev* command as the first argument.

For example help would be done using

```
sslm-mgr dev -h
```

The tools provided :

Tool	Purpose
sslm-auth-hook	internal - used with certbot's manual hook option
sslm-dhparm	generate Diffie Hellman paramater file(s)
sslm-info	display info about cert.pem, csr.pem, chain.pem, privkey.pem, etc
sslm-mgr	primary tool for certificate management
sslm-verify	verifies any cert.pem file using public key from chain.pem

2.2 Groups & Services

To help us organize all the data we introduce groups and services.

What are groups? There are only two kinds of groups: Certificate Authorities and Apex Domains. CA can be self-signed or Letsencrypt et al.

2.2.1 Groups

Certificate Authorities:

The job of a CA is to take a CSR and send back a signed cert.

- Self signed

  self-signed certs use intermediate CA to sign certs. Intermediate CA, in turn, is signed by self signed root CA. Using self signed is a good place to start when getting set up and exploring.

- Letsencrypt

  When comfortable, using their test server, which is more generous with limits, is a good way to prepare for the final version. LE's test server is invoked by using the *-t* option. When all is working as you desire, simply drop the test option and you're ready to go live.

2.2.2 Apex Domains

An Apex domain is the *main* part of the domain that has it's own DNS authority.

If *example.com* has a DNS SOA record, then it would be the apex domain and any subdomain, such as *foo.example.com* would be a part of that apex domain. So, whenever we deal with DNS, we always deal with the apex domain.

2.2.3 Services

Each service gets 1 certificate.

An apex domain may want/need different certs for different services. Each service has one certificate.

An apex domain, for example, may have a mail service and a web service. Each of these has it's own unique cert. Now, mail may use 2 certs, elliptic curve and RSA, then we would simply have 2 services for mail. In this case lets call them *mail-ec* and *mail-rsa* and lets call the web service *web-ec*. Its good to name services in a way thats useful for administrator - it has no significance to the code other than the name must be a good filename so cannot contain / etc.

In the same vein, for self signed CA certs, we have 2 items - a *root* cert and an *intermediate* cert where each belongs the special group *ca*. Again, each of these is a separate service.

Since each service has its own certificate, each has its own X509 name which describe what it is. This includes things like Common Name, Alternative Names and organization. In this case it includes info about the keys to be used and which entity is provides the signed certificate.

Each service has it's information provided by a service file. It has all the information needed to create keys and CSRs as well as certs. This include key type, various *name* fields along with which CA should be used. The *name* fields are essentially *x509* Name[6] fields. These include things like Common Name, Organization and so on.

CSR (certificate signing request) contains the *subject* organiziation (thats the apex domain org) information along with the public key. The private key is kept in a separate file. The CSR is sent to the CA which, all being well, returns a (signed) certificate.

The resulting cert and certificate chain(s) are kept together with the key and CSR files. A cert is signed by the *Issuer* and in addition to the signature contains the public key. The *chain* file contains the public key and x509 Name of the certificate issuer.

There are a couple of tools provided (*sslm-verify* and *sslm-info*) that make it easy to validate a certificate or display information about it. *sslm-info* works on all the *sslm-mgr* outputs : keys, csrs, certs, chains, fullchains and bundles.

2.3 Key/Cert Files

- CSR (certificate signing request)

  Each certificate for is generated from its CSR which contains the public key. Public key is generated from the private key so there is no need to save a public key.

[6] x509 Name https://en.wikipedia.org/wiki/X.509

A CSR is always used make a cert. This provides control as well as consistency across CAs, be they self or other. The public key is in the CSR and also in the certificate provided and signed by the CA. We support both RSA and Elliptic Curve (EC) keys. EC is strongly preferred. In fact, while RSA keys are still used they are only needed by ancient client software for browsers and email. That said, RSA is still in common use for DKIM[7] signing for some reason. We DKIM sign outbound mail with both RSA and EC.

- Cert

  Each cert contains the public key which is signed by the CA. It carries the *subject* apex domain name along with 'subject alternative names' or SANS. SANS allow a certificate to contain multiple domain or subdomain names. The *issuer*, which signed the certificate, has it's name in the cert as well. Name in this context is an X509 name meaning, common name, organization, organization unit and so on.

- Certificate chains

  - **chain** = CA root cert + Signing CA cert

    Signing CA cert is usually the CA Intermediate cart(s) Note that the root cert may or may not be included by CAs other than LE For those client chain = signing ca

  - **fullchain** = Domain cert + chain

  - **bundle** = priv-key + fullchain.

    A bundle is just a chain made of the private key plus the fullchain. This is preferred by postfix[8].

- Private key

  Also called simply the *key*. It is stored in a file with restricted permissions. The companion public key can be generated from the private key. By always generating the public key from the private key, they are guaranteed to remain consistent.

Key, CSR and certificate files are stored in the convenient PEM format. Certificates use X509.V3[9] which provides for *extensions* such as SANS which are critical to have. CSR files use *PKCS#10*[10] which can carry the same set of X509 extensions.

2.4 Tool Commands

As mentioned above, once things are set up for your use case, then all that's needed is periodically run

```
sslm-mgr -renew
```

which will check get new certs, if it's time to renew. A couple of hours later make those certs live by doing:

```
sslm-mgr -roll
```

2.4.1 sslm-mgr

Has 2 modes - a *regular* mode and a developer or *dev* mode. In either case, the groups and services are read from the *ssl-mgr* config file. The config file values *can* be overridden from the command line.

To specify a group and service(s) on the command line use the format:

```
... <group-name>:<service_1>,<service_2>,...
```

[7] DKIM -> https://datatracker.ietf.org/doc/html/rfc6376
[8] Postfix TLS -> https://www.postfix.org/postconf.5.html#smtpd_tls_chain_files
[9] X509 V3 -> https://datatracker.ietf.org/doc/html/rfc5280
[10] PKCS#10 CSR -> https://www.rfc-editor.org/rfc/rfc2986

For example, for a domain with multiple services, you can limit to one or two services using:

```
sslm-mgr -s example.com:mail-ec
sslm-mgr -s example.com:mail-ec,mail-rsa
```

Help command for *sslm-mgr* :

```
sslm-mgr -h
 usage: /usr/bin/sslm-mgr [-h] [-v] [-f] [-r] [-d] [-t] [-n] [-s] [-renew] [-roll]
             [-roll-mins MIN_ROLL_MINS] [-dns] [-clean-keep CLEAN_KEEP] [-clean-all]
             [grps_svcs ...]

 SSL Manager

 positional arguments:
 grps_svcs             List groups/services: grp1:[sv1, sv2,...] grp2:[ALL] ...
                       (default: from config)

 options:
 -h, --help            show this help message and exit
 -v, --verb            More verbose output
 -f, --force           Forces on for renew / roll regardless if too soon
 -r, --reuse           Reuse curr key with renew. tlsa unchanged if using selector=1␣
↪(pubkey)
 -d, --debug           debug mode : print dont do
 -t, --test            Letsencrypt --test-cert
 -n, --dry-run         Letsencrypt --dry-run
 -s, --status          Display cert status. With --verb shows more info
 -renew, --renew       Renew keys/csr/cert keep in next (config renew_expire_days)
 -roll, --roll         Roll Phase : Make next new curr, copy to production, refresh␣
↪dns if needed
 -roll-mins MIN_ROLL_MINS, --min-roll-mins MIN_ROLL_MINS
                         Only roll if next is older than this (config min_roll_mins)
 -dns, --dns-refresh   dns: Use script to sign zones & restart primary (config dns.
↪restart_tool)
 -clean-keep CLEAN_KEEP, --clean-keep CLEAN_KEEP
                         Clean database dirs keeping newest N (see --clean-all)
 -clean-all, --clean-all
                         Clean up all grps/svcs not just active domains

 For dev options add "dev" as 1st argument
```

When more control is needed then *dev* mode offers above commands plus few more options. To see developer help:

```
# sslm-mgr dev -h
 usage: /usr/bin/sslm-mgr ... [-keys] [-csr] [-cert] [-copy] [-ntoc] [-certs-prod]
             [grps_svcs ...]

 SSL Manager Dev Mode

 positional arguments:
 grps_svcs             List groups/services: grp1:[sv1, sv2,...] grp2:[ALL] ...␣
↪(default: see config)

```

(continues on next page)

(continued from previous page)

```
options:
... same as above plus:
-cert, --new-cert     Make new next/cert
-certs-prod, --certs-to-prod
                      Copy keys/certs : (mail, web, tlsa, etc)
-copy, --copy-csr     Copy curr key to next (used by --reuse)
-csr, --new-csr       Make next CSR
-fsr, --force-server-restarts Forces server restarts even if not needed
-keys, --new-keys     Make next new keys
-ntoc, --next-to-curr Move next to curr

For standard options drop "dev" as 1st argument
```

2.5 Config Files

Sample configs are show in Appendix *Appendix* and the files themselves are provided in *examples/conf.d* directory.

When first setting up its a good idea to start with creating a self signed CA and use that. When you're ready then change the signing CA to letsencrypt in the service file and run with the LE test-cert server by using

```
sslm-mgr --test
```

You may also use the letsencrypt *–dry-run* option.

Once that is working for you then you use the normal LE server by dropping the test option.

Config files are located in *conf.d*. There are 2 shared configs and one config for each group/service. Service configs files resides under their *group* directory.

The common configs are *ssl-mgr.conf* and *ca-info.conf* and are used for all groups and services.

ssl-mgr.conf is the main config file and we'll go over it in detail below. It includes the list of domains and their services. If it's needed, the tool can also take 1 or more groups and services on the command line.

ca-info.conf is a list of available CAs. Each CA name can be referenced in service configs to request that CA to provide the certificate.

As described earlier, there are 2 kinds of groups: *CA* and *Domain* groups. The *CA* group is for self created CAs while *domain* are named by the apex domain. Each group item has 1 or more *services*.

Each service gets it's own certificate. Typically services are named for the purpose they are used for (mail, web etc) but also for any characteristics of the certificate, such key type (RSA, Elliptic Curve) and sometimes by the CA as well.

Each (*group*, *service*) pair is described by it's own config located in the file:

```
conf.d/<group>/<service>
```

This file describes the organization and details for one service. This includes Which CA is to sign the certificate as well as any DANE TLS[11] info needed to generate TLSA records.

N.B. Each service is to be signed by the designated CA.

If you want 2 certs signed by 2 different CAs, e.g. both self and letsencrypt, then each would have it's own separate service and associated config file.

E.g. mail-self and mail-le. For each domain, the TLSA records for all services are aggregated into a single file, tlsa.rr to be included by the DNS server.

[11] TLSA https://datatracker.ietf.org/doc/html/rfc6698

N.B.
letsencrypt signing the same CSR counts towards their limits independent of validation method used (http-01 or dns-01).

2.5.1 Service Config

Info for each service to create it's cert. Each domain may have separate certs for different services (mail, web, etc). Each service must therefore have it's own unique config file. Its good practice to use separate certs for each different use cases, to help mitigate any impact of key related security issues.

Each config provides:

- Organization info (CN, O, OU, SAN_Names, …)
- name, org, service (mail, web etc)
- Which CA should will be requested to sign this cert + validation method). Self signed dont need a validation method. + Letsencrypt, for example, allows http-01 and dns-01 as validation methods.
- DANE TLS info - list of (port, usage, selector, match) - e.g. (25,3,1,1)
- Key type for the public/private key pair

2.6 Output

All generated data is kepy in a dated directory under the *db* dir and links are provided for *curr* and *next*

- curr -> db/<date-time>
- next -> db/<date-time>
- prev -> db/<date-time>

After a cert has been successful generated, each dir will contain :

File	What
privkey.pem	private key
csr.pem	certificate signing request
cert.pem	certificate
chain.pem	root + intermediate CA cert
fullchain.pem	cert + chain
bundle.pem	privkey + fullchain
info	Contains date/time when next was rolled to curr (curr only)

The bundle.pem file, which has the priv key, is preferred by postfix to provide atomic udpate and avoid potential race during updates. That could happen if key and cert are read from separate files.

In addition there are the acme challenge files. The *ssl-mgr.conf* file is where to specify where to store these files.

2.6.1 DNS-01 Validation

For dns-01 the location is specified as a directory:

```
[dns]
    acme_dir = '...'
```

The acme challenges will be saved into a file under <*acme_dir*> with apex domain name as suffix:

```
<acme_dir>/acme-challenge.<apex_domain>
```

The format of the DNS resource record is per RFC 8555[12] spec. The challenge file should be included by the DNS zone file for that apex domain. Once the challenge session is complete, the file will be replaced by an empty file, which ensures that there are no errors including it in the domain zone file.

2.6.2 HTTP-01 Validation

For http-01 validation the location is specified by *server_dir* directory:

```
[web]
    server_dir = '...'
```

The individual challenge files, one per (sub)domain will be saved in a file following RFC 8555[13] spec:

```
<server_dir>/<apex_domain>/.well-known/acme-challenge/<token>
```

If the web server is not local then ssh will be used to deliver the file the remote server.

N.B. In all cases please ensure that the process has appropriate write permissions.

2.6.3 DANE-TLSA DNS File

If DANE is on for any service, then the TLSA records will be saved under one or more directories specified in the *[dns]* section of *ssl-mgr.conf*.

```
[dns]
     ...
     tlsa_dirs = [<tlsa_1>, <tlsa_2>, ...]
```

Each directory, *<tlsa_1>*, *<tlsa_2>* etc, will be populated with one file per apex_domain containing the TLSA records for that domain. The file will be named:

N.B. Mail server needs a TLSA record for each key/certificate is used. If, for example, postfix is set up to use either *RSA* or *EC* certs, then you **must** provide a TLSA record for both of them. And there must be record for the apex domain as well as every MX host. We determine the MX hosts via DNS lookup of the apex domain.

```
tlsa.<apex_domain>
```

Each file should be included by the DNS zone file for that apex domain.

2.7 Certbot

A few notes on certbot and how we're using it.

In addition to the database directory (*db*) there is also a *cb* dir which is provided to certbot. Certbot uses to to keep letsencrypt accounts. Each group-service has its own everything - this includes it's own certbot *cb* and thus separately registered LE (Letsencrypt) account for each service.

We are using cerbot in manual mode. This gives us a lot of control and allows us to use our own generated CSR as well as to specify where the resulting cert and chain files get stored.

[12] DNS-01 Acme Challenge URI -> https://datatracker.ietf.org/doc/html/rfc8555#section-8.4
[13] HTTP-01 Acme Challenge URI -> https://datatracker.ietf.org/doc/html/rfc8555#section-8.3

When sending a CSR with apex domain plus sub-domains, each (sub)domain gets a challenge and each challenge must be validated by LE before cert is issued. Challenges can be validated by acme http-01 or dns-01. Wildcard sub-domains (*.example.com) can only be validated using dns-01.

Certbot sends each challenge to a *hook* program. The *hook* program is called once per challenge. Information about the challenge and which sub-domain are passed to the *hook* program in environment variables. Env variables also tell the program how many more challenges remain to be sent. Once all the challenges have been delivered - and only after the *hook* program returns - LE will then seek to validate all of the acme challenges, whether http or dns validation is being used.

This is actually really good - it means that we can push all the challenges out - and wait for every DNS authoritative name server to have the TXT records before allowing the hook to return once it has every acme challenge.

In older versions of certbot, validation took place after each sub-domain challenge, and for DNS that meant dns refresh - wait for NS to udpate - LE checks and sends next challenge. This could potentially very long wait times - I read of some folks waiting many hours. Now with the new way as described above, whether DNS or HTTP challenge, it takes only seconds or minutes.

It seems to me that LE checks directly with each authoritative NS, which is the most efficient way to check - rather than waiting on some random recursive server to get updated.

2.8 TLSA Note

The service config allows DANE to be specified.

The input field takes the form of a list, one item per port:

```
dane_tls = [[25, 'tcp', 3, 1, 1], [...], ...]
```

Each item has port (25 here), the network protocol (tcp) along with *usage* (3), *selector* (1) and *hash_type* (also 1).

You should use (3,1,1).

The dane records normally contain the current TLSA records. During rollover they contain both current and next ones, and after rollover completes, and next becomes current then we're back to the normal case with only current TLSA records.

Each apex domain has it's own file of TLSA records, *tlsa.<apex_domain>*.

The *ssl-mgr.conf* DNS section also specifies where these DNS TLSA record files should be copied to - so that the DNS tools can include them in the apex domain zone file.

The best way to handle the dane resource records is by using $INCLUDE in dns zone file to picks up *tlsa.<apex_domain>* file.

DNS server is refreshed (i.e. zone files signed and primary server is restarted) whenenever a dane tlsa file changes.

The TLSA records change when the private key is updated (leading to change in the hash itself) or when the dane-info is changed (e.g. change of ports or other dane info). It certainly changes after a *renew* builds new keys/certs in *next* and after *roll* when the new *curr* is updated.

For doing rollover properly, order is important.

```
curr →  curr + next →   DNS
```

After 2xTTL or longer:

```
next →  curr →   update mail server →   refresh DNS
```

sslm-mgr takes care of this.

While it is true that reusing a key, means not having to deal with key rolloever as often, that only helps when doing things manually. And in fact even doing it manually, doing things less frequently may mean mistakes are more likely. There is also a small security reduction obviously in reusing a key.

When things are automated, as here with *sslm-mgr* taking care of everything, then there is little benefit to key reuse. So we support it, but we recommend just renew and roll and all will be fine :)

2.9 sslm-mgr application

2.9.1 Usage

To run - go to terminal and use :

```
sslm-mgr --help
```

2.9.2 Configuration

The configuration file for ssl-mgr is chosen by checking for as the first directory containing a *conf.d* directory from the list of *topdir* directories:

```
<SSL_MGR_TOPDIR>
./
/etc/ssl-mgr/
```

Where the directories are checked in order and *<SSL_MGR_TOPDIR>* is an environment variable that can be set to take preference.

The config files are located in *topdir/conf.d/* and certificates, key files and so on are saved under *topdir/certs*.

For example if there environment variable is not set and the directory *./conf.d* exists, then it becomes the top level directory. And all configuration files reside under *./conf.d*.

2.10 Log Files

Logs are found in the log directory specified by the global config variable:

```
[globals]
    ...
    logdir = 'xxx'
```

There are 3 kinds of log files in the log directory.

- *<logdir>/sslm*: General application log
- *<logdir>/cbot*: Application log while interacting with letsencrypt via certbot.
- *<logdir>/letsencrypt/letsencrypt.log.<N>*: Letsencrypt log provided by cerbot.

CHAPTER

THREE

APPENDIX

3.1 Dealing with Possible Problems

Once the configuration files are set up it can be helpful to start with self-signed CA root certificate and a local CA intermediate certificate (signed by that root cert). Use the local intermediate cert to sign your application certificates.

Once this is all working then try using Letsencrypt CA. Upon successful completion the *certs* directory holds all outputs including historical data (older certs and so on).

A subset of the output data is then copied to the production directory. It may also be copied to remote servers if configs request that. After certs are updated and copied then the list of programs to run specified in the *post_copy_cmd* config variable will be run.

If there is a problem for some reason, then updating production will be avoided to minimize any production impact. It is conceivable that after some error conditions, the production directory could get out of sync.

On any subsequent run after experiencing some error condition, *sslm-mgr* will detect whether any production files are out of sync and issue a warning together with a suggestion how to proceeed.

Please note that the very first run of *sslm-mgr* after updating to version *6.1.x* from *6.0.x*, will automatically re-sync production dir.

Manual intervention will be required should errors be detected in any runs after that first one with auto re-sync.

First step is likely to get production in sync. This can be done using the dev option:

```
sslm-mgr dev --force --certs-to-prod
```

to bring production back in sync. You may want to restart the various servers using:

```
sslm-mgr dev --force-server-restarts
```

You may want to renew the certs:

```
sslm-mgr -renew
```

and wait the usual 2-3 hours and then roll as usual:

```
sslm-mgr -roll
```

3.2 Self Signed CA

The *examples/ca-self* directory has sample how to do this. The CA has a self-signed root certificate (*my-root*) along with an intermediate certificate (*my-int*) which is signed by the root cert. Other certs are then signed by the intermediate certificate.

The 2 public CA certs then need to be added to the linux certificate trust store. To do this copy each cert as below and update the trust store:

```
cp certs/ca/my-root/curr/cert.pem /etc/ca-certificates/trust-source/anchors/my-root.
↪pem
cp certs/ca/my-int/curr/cert.pem /etc/ca-certificates/trust-source/anchors/my-int.pem
update-ca-trust
```

Since browsers do not typically use the system certificate store the same certs will need to be imported into each browser. This can be dont manually in the GUI or using *certutil* provided by the *nss* package. Modern browsers typically keep the certificates in a file called *cert9.db* which can be updated using for example something like this (untested):

```
cert9='<path-to>/cert9.db'
cdir=$(dirname $cert9)
certutil -A -n "my-int" -t "TC,C,TC" -i xxx/my-int/curr/cert.pem -d sql:$cdir
```

Please see *certutil* man pages for more info.

3.3 Sample Cron File

```
#
# Renew certs
#  - certs renew (check) every Tue afternoon and roll 3 hours later
#
30 14 * * 2 root /usr/bin/sslm-mgr -renew
30 17 * * 2 root /usr/bin/sslm-mgr -roll

#
# update dh parms:
# will update if existing file is older than min age.
# The default min age is 120 days. Use -a to change min age.
#
30 2 5 * 2 root /usr/bin/sslm-dhparm -s /etc/ssl-mgr/prod-certs
```

3.4 Config ca-info.conf

```
[le-dns]    # Used to sign client certs
    ca_desc = 'Letsencrypt: dns-01 validation'
    ca_type = 'certbot'
    ca_validation = 'dns-01'

[le-http]    # Used to sign client certs
    ca_desc = 'Letsencrypt: http-01 validation'
    ca_type = 'certbot'
    ca_validation = 'http-01'

[my-root] # To sign our own intermediate 'sub' certs
    ca_desc = 'My Self signed root : EC signs my intermediate certs'
    ca_type = 'self'

[my-sub]  # Used to sign client certs
```

(continues on next page)

(continued from previous page)

```
    ca_desc = 'My intermediate : EC signs client certs'
    ca_type = 'local'
```

3.5 Config ssl-mgr.conf

```
[globals]
    verb = true
    sslm_auth_hook = '/usr/lib/ssl-mgr/sslm-auth-hook'      # For certbot
    prod_cert_dir = '/etc/ssl-mgr/prod-certs'
    logdir = '/var/log/ssl-mgr/ssl-mgr/Logs'

    clean_keep = 5
    min_roll_mins = 90
    renew_expire_days = 30

    dns_check_delay = 240
    dns_xtra_ns = ['1.1.1.1', '8.8.8.8', '9.9.9.9', '208.67.222.222']

    post_copy_cmd = [['example.com', '/etc/ssl-mgr/tools/update-permissions'],
                     ['voip.example.com', '/etc/ssl-mgr/tools/voip-checker']
                     ]


#
# Groups & Services
#
[[groups]]
    active=true
    domain='example.net'
    services=['web-ec']

[[groups]]
    active=true
    domain = 'example.com'
    services = ['mail-ec', 'mail-rsa', 'web-ec']

[[groups]]
    active=true
    domain = 'ca'
    services = ['my-root', 'my-sub']

#
# DNS primary provides authorized NS (name servers) and MX hosts of apex_domain
# Must have at least one for acme dns-01
#
[[dns_primary]]
    domain = 'default'
    server = '10.1.2.3'
    port = 11153

[[dns_primary]]
```

(continues on next page)

(continued from previous page)

```
    domain = 'example.com'
    server = '10.1.2.3'
    port = 11153

#
# Servers
#
[dns]
    restart_cmd = '/etc/dns_tools/scripts/resign.sh'
    acme_dir = '/etc/dns_tool/dns/external/staging/zones/include-acme'
    tlsa_dirs = ['/etc/dns_tool/internal/staging/zones/include-tlsa',
                 '/etc/dns_tool/external/staging/zones/include-tlsa',
                 ]

    # restart trigger when dns (TLSA) zones have changed.
    depends = ['dns']

[smtp]
    servers = ['smtp1.internal.example.com', 'smtp2.internal.example.com']
    # If using sni_maps
    #restart_cmd = ['/usr/bin/postmap -F lmdb:/etc/postfix/sni_maps', '/usr/bin/
↪postfix reload']
    restart_cmd = '/usr/bin/postfix reload'
    svc_depends = [['example.com', ['mail-rsa', 'mail-ec']]]
    depends = ['dns']

[imap]
    servers = ['imap.internal.example.com']
    restart_cmd = '/usr/bin/systemctl restart dovecot'
    svc_depends = [['example.com', ['mail-rsa', 'mail-ec']]]

[web]
    servers = ['web.internal.example.com']
    restart_cmd = '/usr/bin/systemctl reload nginx'
    server_dir = '/srv/http/Sites'                  # Used for acme http-01 validation
    svc_depends = [['any', ['web-ec']]]

[other]
    # these servers get copies of certs
    servers = ['backup.internal.example.com', 'voip.internal.example.com']
    restart_cmd = ''
```

3.6 Config Service : example.com/mail-ec

```
#
# example.com : mail-ec
#
name = 'Example.com Mail'
group = 'example.com'
service = 'mail-ec'
```

(continues on next page)

(continued from previous page)

```
#signing_ca = 'my-sub'
#signing_ca = 'le-http'
signing_ca = 'le-dns'
renew_expire_days = 30

# Include tls.example.com in zone file to use
#  => [[port, proto, usage, selector, match], ...]
dane_tls = [[25, 'tcp', 3, 1, 1]]

[KeyOpts]
    ktype = 'ec'
    ec_algo = 'secp384r1'

[X509]
    # X509Name details
    CN = 'example.com'
    O = 'Example Company'
    OU = 'IT Mail'
    L = ''
    ST = ''
    C = 'US'
    email = 'hostmaster@example.com'    # required to register with letsencrypt

    sans = ['example.com', 'smtp.example.com', 'imap.example.com', 'mail.example.com']
```

3.7 Directory tree structure

Directory Structure. By default we only use EC keys, can add RSA if required. We use 'ec' as a label to keep things clear and allow easy way to change to new key types (RSA or other).

Input:

```
conf.d/
    ssl-mgr.conf
    ca-info.conf

    example.com/
        mail-ec
        mail-rsa
        web-ec

    example.net/
        web-ec

    ca/
        my-root
        my-sub
    ...
```

Output - Final Production Certs:

```
prod-certs/
    example.com/
        tlsa.example.com

        dh/
            dh2048.pem
            dh4096.pem
            dhparam.pem -> dh4096.pem
            ...
        mail-ec/
            curr/
                privkey.pem
                csr.pem
                chain.pem
                fullchain.pem
                cert.pem
                bundle.pem
                tlsa.rr
                info
        web-ec/
            ...
        ...
```

Output - Internal Data

```
certs/
    example.com/
        tlsa.example.com

        mail-ec/
            curr -> db/date1
            next -> db/date2

            db/date1/
                csr.pem
                privkey.pem
                cert.pem
                chain.pem
                fullchain.pem
                bundle.pem
                tlsa.rr
            cb/
                [files used by cerbot]

        web-ec/
            curr -> db/date1
            next -> db/date2

            db/date1/
                ...
            cb/
                [files used by cerbot]
```

(continues on next page)

(continued from previous page)

```
        .. other services

    example.net/
        ...
```

3.8 Installation

Available on

- Github
- Archlinux AUR

On Arch you can build using the provided PKGBUILD in the packaging directory or from the AUR. To build manually, clone the repo and :

```
rm -f dist/*
/usr/bin/python -m build --wheel --no-isolation
root_dest="/"
./scripts/do-install $root_dest
```

When running as non-root then set root_dest a user writable directory

3.9 Dependencies

- Run Time :

Package	Comment
python	3.13 or later
dnspython	
cryptography	
dateutil	
lockmgr	Ensures 1 app runs at a time
pyconcurrent	Optional - provides run_prog()

- Building Package:

Package	Comment
git	
hatch	
wheel	
build	
installer	
rsync	
sphinx	Optional (build) docs:
texlive-latexextra	Optional (build) docs aka texlive tools

3.10 Philosophy

We follow the *live at head commit* philosophy. This means we recommend using the latest commit on git master branch. We also provide git tags.

This approach is also taken by Google[1][2].

3.11 License

[1] https://github.com/google/googletest
[2] https://abseil.io/about/philosophy#upgrade-support

CHAPTER

FOUR

CHANGELOG

4.1 Tags

```
2.2.0 (2024-03-29) -> 6.2.0 (2025-07-10)
75 commits.
```

4.2 Commits

- 2025-07-10 : **6.2.0**

```
Make sure all changes to certs/ are properly marked so that production is
updated for any change.
Fix buglet with min_roll_mins option.
Update readme
Update to README
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2025-07-10 : **6.1.0**

```
               *Version 6.1 :*
               * New integrity check.
                 On each run *sslm-mgr* validates that the production directory is up␣
↪to
                 date
                 and consistent with the current suite of certificates, keys and TLSA
                 files.
                 If not, it explains what the problem is and suggests possible ways to
                 proceed.
                 Note that the first run after updating to *6.1* it will
                 automatically re-sync production directory if necessary. No action is
                 required by you.
               * Keep certs and production certs fully synced.
                 Includes removing *next* directory from production after the *roll*
                 has happened and *next* is no longer needed. This change allows us to
                 check
                 that production is correctly synchronized. Earlier versions did not
                 remove any files from production, needed or not.
               * New dev option *--force-server-restarts*.
               * Add ability to specif the top level directory (where configs and␣
↪outputs
```

(continues on next page)

(continued from previous page)

```
                are read from / saved to) via environment variable *SSL_MGR_TOPDIR*.
              * External programs are run using a local copy of *run_prog()* from
                the *pyconcurrent* module.
              You can also install *pyconcurrent* which will ensure the latest
                version is always used.
2025-07-08    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2025-07-08 : **6.0.0**

```
              New **major version 6.0* released. Includes:
                  * PEP-8, PEP-257 and PEP-484 style and type annotations.
                  * Major re-write and tidy ups.
                  * Split up various modules (e.g. certs -> 5 separate crypto␣
↪modules.)
                  * Ensure config and command line options are 100% backward␣
↪compatible.
                  * Improve 2 config values:
                    Background: Local CAs have self-signed a root CA certificate␣
↪which is
                    then used
                    to sign an intermediate CA cert.  The intermediate CA is in turn␣
↪used
                    to sign
                    application certificates.
                    * ca-info.conf: Intermediate local CA entries.
                      * ca_type = "local" is preferred to "self" (NB both work).
                        "self" should still be used for self-signed root CAs where it
                        makes more sense.
                    * CA service config file for self-signed root certificate:
                      *  "signing_ca" = "self" is now preferred to an empty string␣
↪(NB
                      Both work).
                    * These 2 changes are optional but preferred. No other config␣
↪file
                    changes.
                  * Simplify logging code.
2025-03-11    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2025-03-11 : **5.7.1**

```
              After latex update we needed to fix building latex pdf to avoid error
2025-02-28    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2025-02-28 : **5.7.0**

```
              DANE update:
                for port 25 tlsa records are generated for each MX record same as␣
↪always.
                But now, if port is not 25, then TLSA records are for each subdomain␣
↪in
                the x509 SAN domain list.
                There is also a capability to specify this with additional elemein␣
↪in the
```

(continues on next page)

(continued from previous page)

```
                dane_tls item which can be "MX" or "SANS"
              Expand dane tlsa example config file
2025-02-09    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2025-02-09 : **5.6.0**

```
              Less logging to stdout when not verbose. Keep details in log file
              Increase default saved logs to 200k plus 4 backup files
              Avoid double log of cert expiration when renewing.
                Once when checking and again when renewing
              fix: time_to_renew() now returns the expiration string and caller␣
↪chooses to
              log or not
              small logging improvements
2025-01-10    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2025-01-10 : **5.4.0**

```
              Time to cert expiration now shown with more granularity
2024-12-31    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-12-31 : **5.2.0**

```
              Git tags are now signed.
              Add git signing key to Arch Package
              Bump python vers
2024-12-16    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-12-16 : **5.1.0**

```
              Add support for certbot "--preferred-chain" flag in ca-info.conf
                New config for letsencrypt CA : preferred_chain defaults to unset␣
↪(uses LE
                default).
                 e.g. to switch to newer ECC root set: ca_preferred_chain = "ISRG␣
↪Root X2"
2024-12-13    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-12-13 : **5.0.0**

```
              Bug Fix: Its not an error if copy_file(src, dst) when src non-existent.
                => copying to production failed incorrectly if a tlsa file was␣
↪missing
                when none was needed/generated
2024-11-27    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-11-27 : **4.9.0**

```
              Fix typo in dns server when separate server provided for specific␣
↪domain(s)
              conf.d/ssl-mgr.conf - services can now be wildcard services (ALL or *)
                Every file in group directory that is a service config will be␣
↪included as
```

(continues on next page)

(continued from previous page)

```
                service
              add self signed wild card example
              Fix bug with sslm-info not showing IP addresses in SAN
2024-10-21    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-10-21 : **4.5.0**

```
 * New config variable : renew_expire_days_spread (default 0)
   When set to value > 0, renew will happen between expiry_days ±spread
   days.
   Where spread days is randomly drawn from a uniform distribution between
   -spread and spread.
   Using this keeps the average renewal time the same but with multiple
   certificates
   this helps renewals not all fall on same day even if have same
   expiration.
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-10-21 : **4.4.0**

```
              update Docs/Changelog.rst Docs/ssl-mgr.pdf
              use ipaddress instead of netaddr
              Improve messages; more compact
              Some lint picking
2024-08-18    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-08-18 : **4.3.0**

```
                  New config option *post_copy_cmd*
                     For each server getting copies of certs may run this command on
                     machine on which sslm-mgr is running.
                     The command is passed server hostname as an argument.
                     Usage Example: if a server needs a file permission change for an
                     application user to read private key(s).
                     This option is a list of *[server-host, command]* pairs
2024-08-17    update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-08-17 : **4.2.0**

```
              X509v3 Extended Key Usage adds "Time Stamping"
              Changed sslm-dhparm to generate RFC-7919
                 Negotiated Finite Field Diffie-Hellman Ephemeral Parameters files -␣
↪with
                 the default
                 now set to ffdhe8192 instead of ffdhe4096. User options -k␣
↪overrides the
                 default as usual
                 NB push prod certs to all servers using: sslm-mgr dev -certs-prod
                 NB TLSv1.3 restricts DH key exchange to named groups only.
              openssl trusted certificates there is ExtraData after the cert
                 which has the trust data. cryptography.x509 will not load this so␣
↪strip
                 it off.
```

(continues on next page)

(continued from previous page)

```
                see : https://github.com/pyca/cryptography/issues/5242
2024-06-11     update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-06-11 : **4.0.2**

```
Tweak readme
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-06-11 : **4.0.1**

```
Add netaddr as a dependency (used for having IP addresses in alt-names)
Add couple of comments to end of readme about using self-signed certs
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-06-11 : **4.0.0**

```
               Bug fix: CA certs need to be marked as CA and set certificate signing
               ability
2024-05-29     update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-05-29 : **3.7.0**

```
                   Add comment to Readme about new self signed CA example
                   Tweak log message on cert expiration
2024-05-28     update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-05-28 : **3.6.0**

```
                   Skip writing tlsa file if woule be empty.
                   Be more tolerant of missing input
                   Add working example for self signed web server
2024-05-26     update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-05-26 : **3.5.0**

```
bug fix with self signed root cert expiration not using sign_end_days in
config
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-05-26 : **3.4.0**

```
bugfix for self signed cert - fix argument typo
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-05-26 : **3.3.0**

```
               Avoid errors when missing servers
2024-05-22     update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-05-22 : **3.2.4**

```
               README updates
2024-05-21     update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-05-21 : **3.2.2**

```
More readme updates
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-05-21 : **3.2.1**

```
                update readme
2024-05-20      update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-05-20 : **3.2.0**

```
Tweak logging - more info about nameserver checks and visually tidier
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-05-20 : **3.1.1**

```
                Seems possible that letsencrypt dns-01 may not always use the apex␣
↪domain
                    authoritative servers or perhaps their (secondary) check can lag␣
↪more.
                    At least it seems that way lately.
                    We tackle this with the addition of 2 new variables to the top␣
↪level
                    config:
                    See README : dns-check-delay and dns_xtra_ns.
                improve the way nameservers are checked for being up to date with acme
                challenges.
                    First check the primary has all the acme challenge TXT records.␣
↪Then
                    check
                    all nameservers, including the *xtra_ns* have the same serial as␣
↪the
                    primary
                Code improvements and cleanup in dns module.
                buglet whereby the cleanup code was incorrectly calling for dns␣
↪nameserver
                validation.
2024-04-23      update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-04-23 : **2.5.0**

```
                Adjust for upcoming python changes.
                Some argparse options have been deprecated in 3.12 and will be removed␣
↪in
                3.14
2024-04-21      update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-04-21 : **2.4.0**

```
                Enhance non-dns restart_cmd to allow a list of commands. Useful for␣
↪postfix
                when using sni_maps which must be rebuilt to get new certificates
2024-03-29      remove duplicate depends in PKGBUILD
                update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-03-29 : **2.3.1**

```
more little readme changes
minor readme tweak
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-03-29 : **2.3.0**

```
Add PKGBUILD depends : certbot and optdepends: dns_tools
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-03-29 : **2.2.1**

```
update Docs/Changelog.rst
update project version
Fix typo in PKGBUILD
update Docs/Changelog.rst Docs/ssl-mgr.pdf
```

- 2024-03-29 : **2.2.0**

```
update cron sample file comment
Initial Commit
```

CHAPTER

FIVE

MIT LICENSE

CHAPTER

SIX

HOW TO HELP WITH THIS PROJECT

Thank you for your interest in improving this project. This project is open-source under the MIT license.

6.1 Important resources

- Git Repo

6.2 Reporting Bugs or feature requests

Please report bugs on the issue tracker in the git repo. To make the report as useful as possible, please include

- operating system used
- version of python
- explanation of the problem or enhancement request.

6.3 Code Changes

If you make code changes, please update the documentation if it's appropriate.

CHAPTER

SEVEN

CONTRIBUTOR COVENANT CODE OF CONDUCT

7.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

7.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

7.3 Our Responsibilities

Maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

7.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at <arch@sapience.com>. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The Code of Conduct Committee is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

7.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

7.7 Interpretation

The interpretation of this document is at the discretion of the project team.

CHAPTER

EIGHT

INDICES AND TABLES