ssl-mgr

Release 2.3.0

Gene C

Mar 29, 2024

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.
  - Also support *http-01*, but *dns-01* is definitely, strongly preferred
- support for *DANE TLS*

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

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

- **renew - creates the new certificate(s).**
  New ones are always created in *next*; current ones 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.

1.2 Key Features

- Handles new and renew certificates
- Generates key pairs and CSR to provide maximum control
- Supports http-01 and dns-01 acme challenge
- Outputs DNS files for acme DNS-01 authentication as well as TLSA. These files are included by the zone file to make updates straightforward.
- Uses certbot in manual mode to handle communication with letsencrypt, tracking it's accounts etc.
- Processes multiple domains, each can have multiple certs for different purposes if so desired.

1.3 New / Interesting

Recent changes and any important new info goes here.

- This has been in production for some time
- This is the first version made available to community.

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

Furthermore, admin always has the control, should it be needed, to do whatever they choose.

e.g. a 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 the same set of files.

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.

1.4.2 Diffie-Hellman Parameters

There is an additional tool provided, *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.

The new, preferred and now default DH parameters are based on RFC-7919 rfc_7919 pre-defined named groups. The default is *ffdhe4096*, which only need to be generated once and will only be generated when absent. Strictly these don't need to be in cron, but its convenient to have the program check and create the DH parameters should they be missing. May happen occasionally when 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, just 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.

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 the new keys. 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 live 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. The companion *dns_tools* package takes care of all our DNSSEC needs[4]:

And I recommend using it to simplify the DNS refresh needed TLSA and for validating with Letsencrypt using *DNS-01*. A DNS refresh means resign zones and restart the primary dns server.

[3] acme-challenge : https://letsencrypt.org/docs/challenge-types/
[4] dns_tools : https://github.com/gene-git/dns_tools

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.

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 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

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
ssl-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 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.

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.

2.2.2 Services

Each service has 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 our case this also 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 file. The CSR is sent to the CA and it 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 domain 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.

  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.

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

- Cert

  This cert contains the public key and 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 (Intermediate(s) usually). root may or may not be included by CAs other than LE i.e. client chain = signing ca fullchain

  - fullchain

    Domain cert + chain

  - bundle

    priv-key + fullchain. This is preferred by postfix.

- 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[7] which provides for *extensions* such as SANS which are critical to have. CSR files use *PKCS#10*[8] 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 *dev* mode. For all commands, the groups and services are read from the *ssl-mgr* config file, but *can* also be provided on the command line.

The help for this is:

```
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
```

(continues on next page)

[7] X509 V3 -> https://datatracker.ietf.org/doc/html/rfc5280
[8] PKCS#10 CSR -> https://www.rfc-editor.org/rfc/rfc2986

(continued from previous page)

```
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:

```
# 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)

options:
... same as above plus:
-keys, --new-keys     Make next new keys
-csr, --new-csr       Make next CSR
-cert, --new-cert     Make new next/cert
-copy, --copy-csr     Copy curr key to next (used by --reuse)
-ntoc, --next-to-curr Move next to curr
-certs-prod, --certs-to-prod
                      Copy keys/certs : (mail, web, tlsa, etc)

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

2.5 Config Files

Examples of configs are show in Appendix *Appendix* and the files themselves are in *conf.d/examples*.

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

```
sslm-mgr -t
```

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 common configs and one 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[9] 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.

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**

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

  - 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.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.

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.

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 ssl-mgr application

2.9.1 Usage

To run - go to terminal and use :

```
ssl-mgr --help
```

2.9.2 Configuration

The configuration file for ssl-mgr is …

```
/etc/ssl-mgr/config
```

2.9.3 Options

Available options for ssl-mgr are .. This section can be referenced by *ssl-mgr*

2.10 Log files

Logs are found:

```
${HOME}/log/ssl-mgr
```

2.11 Another Section

More stuff.

CHAPTER

THREE

APPENDIX

3.1 Sample Cron File

```
#
# Renew certs
#  - avoid dnsec key rolls times
#    dns_tools uses locking so just nice not to overlap
#    dnssec renews on 2nd of every month at 8 am and rolls 10 am
#  - certs renew (check) every Tue afternoon and roll 2 hours later
#
30 14 * * 2 root /usr/bin/sslm-mgr -renew
30 16 * * 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.
# Update early morning ahead of any cert renewal.
#
30 2 5 * 2 root /usr/bin/sslm-dhparm -s /etc/ssl-mgr/prod-certs
```

3.2 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
    ca_desc = 'My intermediate : EC signs client certs'
    ca_type = 'self'
```

3.3 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 = 10
    min_roll_mins = 90
    renew_expire_days = 30

#
# 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 = 10053

[[dns_primary]]
    domain = 'example.com'
    server = '10.1.2.3'
    port = 10053

#
# Servers
#
[dns]
    restart_tool = '/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']
```

(continues on next page)

(continued from previous page)

```
[smtp]
    servers = ['srv8.prv.sapience.com', 'srv7.prv.sapience.com']
    restart_cmd = '/usr/bin/systemctl restart postfix'
    svc_depends = [['sapience.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.sapience.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 = ''
```

3.4 Config Service : example.com/mail-ec

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

#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
```

(continues on next page)

(continued from previous page)

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

3.5 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]

        .. other services

    example.net/
        ...
```

3.6 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

3.7 Dependencies

- Run Time :

Package	Comment
python	3.11 or later
dnspython	
cryptography	
dateutil	
netaddr	
lockmgr	Ensures only 1 app runs at a time

- Building Package:

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

3.8 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].

3.9 License

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

CHAPTER

FOUR

CHANGELOG

[2.3.0] —— 2024-03-29

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

[2.2.1] —— 2024-03-29

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

[2.2.0] —— 2024-03-29

```
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

- genindex
- modindex
- search