MARWAY
POWER SOLUTIONS

RCM Technical Note — SSL/TLS Certificates

How to Use Self-Signed and Public CA Certificates for RCM v2.2+

How to Generate and Use Private CA Certificates for RCM v2.2+

Sept 2019 • r3

Marway Power Solutions
1721 S. Grand Ave., Santa Ana, CA 92705
800-462-7929 • marway@marway.com

Table of Contents

HTTPS, TLS, SSL in RCM 2.2 and Later Software

Introduction to Certificates

When using the web browser to operate an RCM power distribution unit, the RCM software supports encrypted traffic—that is the actions and data transferred between the PDU and your web browser are encypted for improved security. The encryption is enabled, in part, through the use of what is called a digital certificate, or just certificate for short ("cert" is also frequently used).

A certificate is an electronic file with a little information about the device it is running on, the organization that owns the device, and some technical details about the configuration of the cryptography which will be used. A certificate can be auto-generated by the PDU, or it can be a process of manually creating the starting point of a certificate which is sent to a special organization called a Certificate authority which completes the certificate.

The auto-generated (also referred to as a self-signed) type of certificate will activate encryption, but there are a number of user-friendliness challenges with this type. A more complete type is a certificate-authority-signed certificate. These are more complicated to create, but they provide better security, and a better user experience.

While not a complete reference for certificates or certificate management, this document aims to introduce enough material to provide the novice a starting point from which to explore additional materials, as well as provide a more experienced person with all the technical details needed to take advantage of the RCM software's capabilities in HTTPS configuration.

RCM HTTPS Fundamentals

RCM software supports HTTPS with SSL 3.0 and TLS 1.2. The underlying NET+OS operating system has limited options for ciphers and hashes. During an HTTPS negotiation, the NET+OS server hello will offer the following options:

- Ciphers:
  - TLS_RSA_WITH_AES_128_CBC_SHA — 0x2F
  - TLS_RSA_WITH_AES_256_CBC_SHA — 0x35
- Key exchange is RSA, length is 2048 bits, hash is SHA256 (version 2.1 and earlier used MD5).

Self-signed and certificate-authority-signed ("CA-signed") certificates can be used. CA-signed certificates can be signed through a third-party certificate authority, or through a private certificate authority. This document explores using all these options.

Self-Signed Certificate

Enabling Self-Signed Certificate for HTTPS

RCM software allows for an auto-generated, self-signed certificate to use TLS for HTTPS. To enable this capability requires selecting HTTPS as the enabled protocol, and in versions 2.2.0 and later, choosing a Certificate Type of Self-Signed. (For versions 2.1 and earlier, self-signed was the assumed and only option.)

Using the web interface

- Navigate to the Network settings page.
- In the HTTP and HTTPS section:
  - Select the Enable Mode of HTTPS.
  - Select the Certificate Type of Self-Signed.
- Restart the PDU software (use System > System Restart button).
- Reconnect to the PDU's address, making sure you use https:// instead of http://.

Using the command line

- View the current settings with the command `#> getHttp`
- Change the `Mode` setting with: `#> setHttp mode https`
- Change the `Mode` setting with: `#> setHttp cert_type self_signed`
- Restart the PDU with: `#> restart`

Challenges with Self-Signed Certificates

Using the auto-generated, self-signed option is the *simplest* way to enable encryption for the web interface, but, depending on your user environment, there may be drawbacks. Modern browsers are more aggressive in displaying "warning walls" when first connecting to devices with self-signed certificates. This will require the user to understand these warnings, know how to verify the connection authenticity, and flag the connection as trusted. For non-technical users unfamiliar with how HTTPS works, this is not the most friendly process to go through—especially if there are multiple PDUs to connect to.

Who Should Use Self-Signed Certificates

In our view, if there's very few nodes (users and pdus) on a private LAN, and users are familiar with how these certificates work (or are prepared to learn), then working through the browser's warnings in order to have it "trust" the self-signed certificate is more efficient than working with CA-signed certificates.

If there's a private LAN and there are a larger number of nodes, it is probably a better option to adopt the use of CA-signed certificates. Either private or third-party signing would be suitable.

If the PDU is to be used across the public internet, use third-party CA-signed certificates. Do not use self-signed certificates.

CA-Signed Certificates

As of RCM 2.2, CA-signed certificates can be used. These certificates can be provided by third party certificate authorities, or by what's known as a private certificate authority—essentially, you (your organization).

By having a certificate validated through a Certificate authority ("CA"), browsers won't present the privacy warnings seen when using self-signed certificates, and of course, there's the advantage of a higher degree of trust that the device being connected to is actually what it claims to be. This is especially important if the PDU will be accessed across the public internet.

The Role of the CA

A certificate authority is an organization responsible for providing signed certificates. But, let's back up and consider the purposes of SSL/TLS certificates. First, they are used to facilitate encryption between two devices. Second, they are used to infuse some trust that the system you are connecting to is what it claims to be. This second purpose is where the Certificate authority's role matters.

Creating certificates is intended to be a two-part process. The first part involves a requestor who needs to serve trusted content. The requestor requests that a certificate be issued which in effect claims "I am ABC." The second part involves a trusted party to validate that the requestor's claim is legitimate. This job is what the CA does—verifies the identity of ABC, and that the requestor has the right to have a certificate which claims it is ABC. When the CA confirms this, it "signs" the certificate request from the requestor (we'll look at this closer below). The completed certificate now has some digital data which operating systems and browsers know means the certificate has some level of trustworthines. The presence or lack of this data in an HTTPS connection is used by the browser to indicate whether the system being connected to is trustable.

How do we trust the CA's? The short answer is that while the top of the food chain used to be the U.S. government, since 2012, it is has been managaged by an industry consortium. Technically, there is nothing stopping anyone from trying to be a CA. But, in order to be effective, a CA's own master certificate must be installed in computers and servers and other devices. Operating system vendors and browser vendors pre-install the master ("root") certificates of the major CA organizations they trust. Today, the standards and policies for CAs is managed by https://cabforum.org/.

Third-Party vs. Private CA Certificates

Knowing the role of the CA is to verify identity, let's compare self-signed, private signed, and third-party signed certificates. With a self-signed certificate, the requestor performs both steps—the request and the signing. Obviously, there could be some trust issues with that. If you or a co-worker did the signing for certificates used on devices only you will connect to, well, that's fine. You could easily inform a handful of other people the certificate is OK, and to go ahead and trust it. Trying to do that for 50 or 100 or more people? Even if they're on the same LAN, that gets to be a burden.

With a third-party-signed certificate, that third party is doing the signing. Since that certificate is connected to a public CA, and that CA's own root certificate is, in theory, widely distributed across operating systems, then there is wide public trust in the identity of that certificate. This makes it easy for almost anyone connecting to the device with that certificate to trust the device actually belongs to who it claims to belong to. (Trusting what is *does* is a whole other matter.)

In between these two is the private Certificate authority. In this scenario, an organization becomes its own Certificate authority to create certificates for devices on its own private network. It will create a root certificate for

itself, and then all internal requests for certificates will be signed using that root. There could be multiple reasons to choose this path, but they generally boil down to control (presumably for better security) and cost.

There is no significant technical differences between private and third-party CA signed certificates. The primary difference between them is intended use, perspective of convenience vs. control, and potentially cost.

Back to the RCM products, from a practical standpoint, if there are users who have to connect to one or more PDUs across the public internet, then in almost every case it is better to use third-party, CA-signed certificates. There's a very small set of circumstances where you'd want to use private certificates. If you don't already know you want to use private CA-signing, and why you want to do it, then you can assume you shouldn't use it. Depending on the type of certificates (DV, OV, EV) and vendor you choose for to use, there may be some cost associated with acquiring third-party CA-signed certificates. There are also good free options.

On the flip side, if all your users will connect to a PDU directly on a local network, or remotely through a VPN, but never remotely over the public internet, then your organization may want to use private CA-signed certificates—especially if it is already accustomed to managing internal infrastructure this way.

Levels of Verification with Signed Certificates

When we consider that a key role of the public certificate authorities is to verify that a requestor is who it says it is, you have to wonder what they actually do to verify this. This can be done with a person working for the third-party CA actually researching and verifying your claims, or it can be done with automation. The former of course can be far more thorough, but there is a cost. The latter would be less thorough, but good enough for many cases, and is done for free by some organizations, or for a comparatively low cost.

Reflecting the different levels of thorougness with which verification can be established, these days there are actually three commonly distinguished types of certificates named domain validated (DV), organization validated (OV), and extended validation (EV).

Domain validation relies on relatively simple verification that the requestor has control over the domain in question. This is easily automated typically through emails. The certificate will be signed by the CA, but the certificate will not include details of the certificate ownership.

Organization validation involves a deeper investigation to verify that the organization identity claimed in the request is valid. The certificate will be signed by the CA, and this time, the certificate *will* include the name of the organization as well, which end users can see, thereby establishing a bit more trust in who they're connecting to.

Extended validation involves the even more effort expended by a CA involving the submission of documents to the CA which they can verify. The certificate is signed, the company information included in the details, and extra digital settings flagged so that the browser can provide an "extra good" indicator for trust.

With a public CA, you can select from any of these types. For private CAs, DV and OV types can be created, but not EV types. To create an OV type, the certificate simply includes the company name and contact info in the certificate details during the signing step.

Creating a Certificate Request

Regardless of whether you use a third-party CA, or act as your own private CA, the process of creating CA-signed certificates is the same.

Use a GUI utility or the OpenSSL command line tools to create a certificate signing request ("CSR"). Certificates for RCM software should following these parameters:

- Key length is 2048
- Key hash is SHA256
- Key format is PEM
- Final certificate format is PEM
- CA's certificate format is PEM

A starting baseline OpenSSL command to generate a CSR would be:

```
openssl req \
    -newkey rsa:2048 \
    -sha256 \
    -keyout trusted_rcm_key.pem \
    -keyform PEM \
    -out rcm_originating_csr.pem \
    -outform PEM \
    -nodes
```

However, this doesn't specify `x509v3` extensions which should be included, particularly for `subjectAltName`, nor does it specify a number of possible policies and configurations which may be prudent for security within your environment. If you have someone to ask, be sure to do so. If you're the one to ask, and you're not sure what this means, then at least research how to include `subjectAltName` in CSRs. This can be done using configuration files or command line options. (More complete examples of commands and configurations are shown in Appendix A: OpenSSL Config File.)

When the command is executed, there will be some output text, and you will be prompted by questions which look like those shown below. For each line, press return to accept the default value shown in the brackets (which you can determine in a configuration file), or enter a new value—which you must enter for the Common Name field. For reference, these are known as the Distinguished Name fields.

```
Country Name (2 letter code) [US]:
State or Province Name (full name) []:
Locality Name (e.g. city) []:
Organization Name (e.g. company) []:              <=== The full legal name of the company
Organizational Unit Name (e.g. division) []:
Common Name (e.g. IP or FQDN) []:192.168.10.100   <=== Or DNS name if you're using them.
Email Address [admin@example.com]:
```

This will generate two files. The trusted_rcm_key.pem file should never be given to anyone not responsible for managing certificates in your organization. It is a secret critical to the security of the certificate. You do *not* send it to a public third-party certificate authority. You might send it to an internal private CA since that would be your own organization managing copies of all internal certificate components. Note that the RCM software does not not have a mechanism to use a passphrase protected keys. The `-nodes` option has openssl skip the addition of a passphrase. If you choose to have passphrases so master copies of the keys are safer, the copy of the key installed in the RCM unit will need to have that removed. (See Appendix A for miscellaneous utility commands.)

The rcm_originating_csr.pem file is what gets sent to the certificate authority. They may require that the file name be something else—maybe something as simple as csr.pem. The name of the file is not critical, it is OK to rename it, or to duplicate it and rename the duplicate.

Returned Signed Files from the CA

Ultimately, the CA organization from whom you're acquiring certificates (public or private) will send back two files. One will be the RCM unit's certificate. This will be the signed version of the originating CSR file. The other file will be the CA company's own certificate. You should specify that you need these files to be in the PEM format. If that's not how they are provided, use either the openssl command line program or another utility to convert them.

It is common for these files from the CA to have very simple names like server.crt and ca.crt or something similar. One of them should be identified with "ca" one way or another so you can tell them apart. Once you have these files, they must be renamed to work in the RCM product.

- Rename the CA's file to trusted_ca_cert.pem
- Rename the server's file to trusted_rcm_cert.pem

Installing CA-Signed Certificate Files

Be sure that the Network settings have HTTPS enabled, and that the Certificate Type is CA-Signed. You can then use FTP to upload the following files in the RCM's file system at /FLASH0/System/.

- trusted_ca_cert.pem
- trusted_rcm_cert.pem
- trusted_rcm_key.pem

After the uploads are complete, restart the RCM software.

Using OpenSSL to Be a Simple Private CA

This is by no means a complete guide on how to be a private CA. There is no doubt that many details which would be best practices are not being addressed. Our goal here is to provide a guide in the very basics of how to use OpenSSL to create private CA certificates. Our assumption is that if you have a small, controlled environment where there is no other private CA authority to rely on, this guide can be used to create CA-Signed certificates which provide some security and user-friendly advantages over the RCM's own auto-generated, self-signed certificates. As with any guides, you may need to adapt the details provided here to the specifics of your computing environment.

OpenSSL

This guide is based on the openssl command line program—you'll need to have that installed. (In theory, newer is better, but the commands and options we'll use go back to versions at least a few years old.) There are GUI applications which do the same things, and our command line details should help you through the GUI options as well if you choose to go that route. Also, libreSSL and other OpenSSL replacements which are forks of OpenSSL should work as well. We'll let the reader find and follow an installation guide for your own operating system and chosen application.

Since we're using a CLI implementation of openssl, there are a few supporting CLI commands we use for file system preparation. These assume `bash` in a POSIX environment. If you're using Windows, you'll need a POSIX substitute (like cygwin) or translate as needed.

Get Organized

You're going to need some file space to store your master CA files, incoming CSRs, and signed certificates. You can decide what to keep copies of, how you want to organize them, and what you want to name them. At a minimum, we're going to lay out a few files so we have some fixed names to call things in the tutorial. Once you've experimented a little, you can decide whether to rename things.

The scope of our Private CA concern is just for Marway's RCM PDUs. Therefore, we're going to create a root folder named `/Marway_RCM_Certs`. So, `cd` into whatever root you want to use for certs, then run these commands:

```
$ cd ~/
$ mkdir Marway_RCM_Certs
$ cd ~/Marway_RCM_Certs
$ mkdir RCM_CA_Masters
$ cd RCM_CA_Masters
$ mkdir Signed_Certs
$ touch index.txt
$ cd ~/Marway_RCM_Certs
```

Create the Config File

If you have only this PDF and not a separate config file, copy the non-comment text of the configuration file from Appendix A. Paste it into a text editor (not a word processor), and save that file as /Marway_RCM_Certs/rcm_ca.cnf. As of this writing, a copy of the whole text file can be found at www.marway.com/software/other — look for resources on the topic of SSL/TLS Certificates. Note that you have to update a few lines in the example configuration file. Search for USRCFG to find the lines.

Create CA Root Key and Certificate

Having created the file structure and configuration file above, we can now get started working on certificates. A CA's root certificate is actually a self-signed certificate. We do this in two steps. First we create a CSR. This command must be entered all on one line. (The \ symbol shows that what follows really should not be on a separate line, it's just show on separate lines here because of space limitations. This format also helps to clearly show you the arguments and values being used.)

```
openssl req \
    -config rcm_ca.cnf \
    -newkey rsa:2048 \
    -keyout rcm_ca_root_key.pem \
    -out rcm_ca_root_csr.pem \
    -outform PEM \
    -nodes
```

When the command is executed, there will be some output text, and you will be prompted by questions which look like those shown below. For each line, press return to accept the default value shown in the brackets (which you control in the rcm_ca.cfg file), or enter a new value—which you must enter for the Common Name field. (For reference, these are known as the Distinguished Name fields.)

CRITCAL NOTE: For convenience purposes in this tutorial, the command as shown does not password protect the private key file (due to the -nodes option). If anyone ever gets ahold of that key file, they can forge certificates. After creating the key, You can protect that key by adding a passphrase to it using a command like:

```
openssl rsa \
    -aes256 \
    -in [key_name] \
    -out [new_key_name]
```

You'll be asked to enter your password. After protecting the file like this, every time it is referenced for use by an openssl command, you'll have to enter the password.

```
Country Name (2 letter code) [US]:
State or Province Name (full name) []:
Locality Name (e.g. city) []:
Organization Name (e.g. company) [Example Company]:
Organizational Unit Name (e.g. division) [PDU Certs]:
Common Name (e.g. IP or FQDN) []:192.168.10.100   <=== THIS WILL BE UNIQUE FOR EVERY PDU
Email Address [admin@example.com]:
```

The key and csr files will be created in /Marway_RCM_Certs. Next, sign the CSR with this command.

```
openssl ca \
    -config rcm_ca.cnf \
    -create_serial \
    -selfsign \
    -in rcm_ca_root_csr.pem \
    -out rcm_ca_root_cert.pem \
    -keyfile rcm_ca_root_key.pem \
    -extensions rcm_ca_v3
```

You should now have three files:

- CSR file: rcm_ca_root_csr.pem
- Certificate: rcm_ca_root_cert.pem
- Private Key: rcm_ca_root_key.pem

Duplicate the file rcm_ca_root_cert.pem, and rename it as trusted_ca_cert.pem. We'll use this later. Next, move the three files listed above into the /RCM_CA_Masters folder. This simply helps organize all the CA master files in one place. From now on, the /Marway_RCM_Certs folder becomes the workspace for all new CSRs and signing.

You are now ready to receive CSRs on behalf of RCM PDUs and generate signed certificates. But your workstations which will connect to those PDUs still won't trust your CA authority. So, we have to solve that first.

Install Your CA Certificate in Computer Workstations

Any computer workstation or mobile device which is expected to connect to an RCM PDU over HTTPS, has to have the rcm_ca_root_cert.pem file installed. The steps to do this are different for each operating system. Another challenging factor is that different operating systems may require the certificate to be in different formats. One common need is to have the PEM format converted to DER. This is how you would do the conversion:

```
openssl x509 -outform der -in rcm_ca_root_cert.pem -out rcm_ca_root_cert.der
```

This guide will rely on you searching the internet for guides on "how to install root certificate on [YOUR_OS]." With desktop GUI OSes you can usually double click the certificate file to launch a utility which gets you started. Use the internet to discover the next set of steps.

Create a CSR for an RCM PDU

Each RCM PDU needs to have its own certificate. For this tutorial we'll assume that each PDU is running a fixed IP address (and not DNS names). Since the PDU cannot be used to generate a CSR, you'll use your workstation to create the CSRs for each PDU like this:

```
openssl req \
    -config rcm_ca.cnf \
    -newkey rsa:2048 \
    -keyout trusted_rcm_key.pem \
    -keyform PEM \
    -out rcm_originating_csr.pem \
    -outform PEM \
    -extensions rcm_csr_v3 \
    -nodes
```

You will be prompted by a series of questions to populate the distinguished name fields (country, organization, common name, etc.). Based on the configuration file (a section named `[rcm_signing_policy]`), some fields will be required or optional, and some may even have to exactly match the entries used for the CA certificate. In response to the "Common Name (e.g. IP or FQDN)" question, enter the IP address of the PDU (or the DNS name if you're using them).

This creates the two files listed in the command. Again, the keyfile is not passphrase protected, and this is now more on purpose. This keyfile will be uploaded to the RCM PDU, and the RCM software cannot read a passphrase protected key file. Therefore, a non-passphrase protected file needs to exist. (You can create a password protected copy of the file for backup if you wish.)

Sign the CSR for an RCM PDU

Now that you are your own certificate authority, you get to sign your own CSRs. Even though you are signing the CSR *yourself*, this is not the same as a self-signed certificate. In that term, "self" is not a reference to who is doing the work, but rather a reference to the one certificate being used for the signing step of the same certificate. As a CA, you will be using the CA certificate to sign incoming CSR certificates for the RCM PDUs.

Edit the cfg File subjectAltName

The first step before we run the signing command is to adjust the configuration file's line which defines the `subjectAltName = IP:` value. Managing this particular detail of the signing process is probably the trickiest. For this tutorial we're using the simplest means to handle this, which is to specify it manually in the configuration file. For creating a handful of certificates, this isn't a big problem (except for remembering to do it). If you have a lot of CSRs to sign, this is not convenient. Giving examples of how to automate this is out of the scope of this guide—you'll need to take advantage of whatever scripting language you're familiar with to work out some improvements.

Run the Signing Command

The following command signs the CSR certificate.

```
openssl ca \
    -config rcm_ca.cnf \
    -create_serial \
    -in rcm_originating_csr.pem \
    -out trusted_rcm_cert.pem
```

This will generate a new file with the name specified by the `-out` argument.

Installing Private CA Certificate Files

Your CA certificate must be installed on all the workstations which will connect to the PDUs with HTTPS. See the section Install Your CA Certificate in Computer Workstations for that discussion.

Be sure that the Network settings have HTTPS enabled, and that the Certificate Type is CA-Signed. You can then use FTP to upload the following files in the RCM's file system at /FLASH0/System/.

- trusted_ca_cert.pem
- trusted_rcm_cert.pem
- trusted_rcm_key.pem

After the uploads are complete, restart the RCM software.

Long Term File Management Considerations

Storing Copies of Files

At this point you need to consider your needs and preferences for managing these files over the long term. One question to consider is once files have been uploaded, do you *need* to keep a copy of all the files created for each PDU? Below are some notes about each of the relevant files.

- trusted_ca_cert.pem — This is the duplicated CA cert created earlier. This gets uploaded to every PDU which is to run HTTPS, and every workstation which is to connect to those PDUs. Keeping a copy of this in the same location as where the CSR files are generated and signed is assumed to be handy for creating a file set to bundle for uploading to the PDU. Move it to wherever you want it to be.
- rcm_originating_csr.pem — This is from the CSR command, and is unchanged by the signing step. It has no secret data, and after the CSR has been signed, there is really no need to keep this file. You can if you want for record keeping, but there's no technical need for it.
- trusted_rcm_key.pem — This was created with the CSR command. This gets uploaded as is to the specific PDU for which the CSR was generated. This is a critical secret data file. By default, our command did not password protect it (the PDU needs it that way). If you want to keep a copy of this file, you might want to create a password protected copy (of course then you need a secure system for storing passwords). You might want to keep a copy so it can be restored to a PDU along with the signed certificate for some reason. However, it may be easier to recreate a new key/cert if the need arsies rather than be concerned about managing copies of these files. Your own organization policies will dictate whether you have a records keeping need here.
- trusted_rcm_cert.pem — This is the new signed certificate. This gets uploaded as is to the specific PDU for which the CSR was generated. This is not a secret file. As discused in the next section, the openssl system is already keeping backups of these files, but you may want your own copies to keep in file bundles organized by PDU addresses.

The Signed Certificate Database

So that you understand more of what's happening behind the scenes, let's have a look inside the `/RCM_CA_Masters` folder. You should see your CA csr, key, and cert files which we created and moved earlier. You should also see an index.txt file and a /Signed_Certs folder. There are others which are not necessary to dig into. These are created by the `openssl ca` commands which generate a database to store certificate backups.

The `index.txt` file keeps a list of every certificate which was signed. The list includes a serial number (3rd column) and an abbreviated form of the distinguished name (company, commaon name, email address, etc.).

The /Signed_Certs folder contains a copy of the signed certificate named with the serial number.

Revocation of Certificates

One topic not addressed by our configuration and command examples is the area of revoking a certificate. Depending on the size of your deployment, it may be a practical concern to worry about certificates which should be deemed invalid before their expiration date. Implementing this can be complex. We don't aim to address it in this guide other than to make you ware of it being something to look into and think about.

Appendix A: OpenSSL Config File

The content below is an openssl configuration file, and includes a bunch of notes for how to use it. Put these contents in the file path ~/Marway_RCM_Certs/rcm_ca.cnf. As of this writing, you can find a downloadable text file of this at www.marway.com/software/other.

```
#==============================================================================
#
# HOW TO CREATE A PRIVATE CERTIFICATE AUTHORITY FOR MARWAY RCM PDUs
#
#==============================================================================

# This file is a sample SSL configuration file set up help create a private
# certificate authority for using HTTPS with Marway RCM power distribution
# units.

# To use this file with the openssl commands discussed below:
# - perform the bash commands in the section Create a CA Certificate Database
# - rename this file to rcm_ca.cfg, and put it in /Marway_RCM_Certs
# - search for the text USRCFG to find lines which you must edit

# You should consult the full documentation at www.marway.com/software
# Look for the PDF download named 'RCM Technical Note - SSL/TLS Certificates'


#------------------------------------------------------------------------------
#
# Create a CA Certificate Database
#
#------------------------------------------------------------------------------

# cd into whatever root you want to use for certs, then run these:
# $ cd ~/
# $ mkdir Marway_RCM_Certs
# $ cd ~/Marway_RCM_Certs
# $ mkdir RCM_CA_Masters
# $ cd RCM_CA_Masters
# $ mkdir Signed_Certs
# $ touch index.txt
# $ cd ~/Marway_RCM_Certs
```

```
#-------------------------------------------------------------------------------
#
# Example Commands
#
#-------------------------------------------------------------------------------

# CREATE THE ROOT CA CSR:
#   openssl req \
#       -config rcm_ca.cnf \
#       -newkey rsa:2048 \
#       -keyout rcm_ca_root_key.pem \
#       -out rcm_ca_root_csr.pem \
#       -outform PEM \
#       -nodes

# SIGN THE ROOT CSR:
#   openssl ca \
#       -config rcm_ca.cnf \
#       -create_serial \
#       -selfsign \
#       -in rcm_ca_root_csr.pem \
#       -out rcm_ca_root_cert.pem \
#       -keyfile rcm_ca_root_key.pem \
#       -extensions rcm_ca_v3
#
#   Duplicate the file rcm_ca_root_cert.pem, rename it as
#   trusted_ca_cert.pem, and upload that to every Marway RCM power
#   distribution unit's /System folder (using FTP).
#
#   This file must also be installed and configured to be a trusted
#   certificate in every workstation which is to connect to any of the
#   RCM PDUs using HTTPS.

# CREATE A CSR CERTIFICATE FOR EACH PDU:
#   openssl req \
#       -config rcm_ca.cnf \
#       -newkey rsa:2048 \
#       -keyout trusted_rcm_key.pem \
#       -keyform PEM \
#       -out rcm_originating_csr.pem \
#       -outform PEM \
#       -extensions rcm_csr_v3 \
#       -nodes
#
#   In response to the multiple questions, make sure to follow the "match"
#   conditions defined in [rcm_signing_policy]. Also, in response to the
#   "Common Name (e.g. IP or FQDN)" question, enter the IP address of the
#   PDU (or the DNS name if you're using DNS names for PDUs).
#
#   This will create two as notes in the -out and -keyout options.
#   The key file can be copied to the specific RCM unit as is, but the
#   csr file must be processed by the next step.
```

```
# SIGN A CSR CERTIFICATE FOR EACH PDU:
#
#   First, edit the line which start with "subjectAltName = IP:" and set it
#   to the exact IP address just defined in the above CSR cert.
#
#   openssl ca \
#       -config rcm_ca.cnf \
#       -create_serial \
#       -in rcm_originating_csr.pem \
#       -out trusted_rcm_cert.pem

# UPLOAD THE FILES TO THE RCM UNIT:
#
#   These files get uploaded to the RCM unit's /System folder via FTP.
#   - trusted_rcm_key.pem -- private key specific to each unit.
#   - trusted_rcm_cert.pem -- signed cert specific to each unit.
#   - trusted_ca_cert.pem -- CA cert common to all RCM units.

# VIEW CERT FILES using:
#   For signed certs:  openssl x509 -text -noout -in {FILENAME}
#   For csr certs:     openssl req -text -noout -in {FILENAME}



#============================================================================
#
# SSL Configurations
#
# Pieces and styles have been adopted from a number of examples.
# However, in the end, https://www.phildev.net/ssl/ was the most
# influential source in getting this example operational.
#
#============================================================================


#----------------------------------------------------------------------------
#
# CA -- configs which apply to using the `openssl ca` command.
#
#----------------------------------------------------------------------------

[ ca ]

default_ca = marway_rcm_ca
```

```
[ marway_rcm_ca ]

# Our own variables for the path. These are not openssl key words.
# ************************************************************
# USRCFG: ** YOU SHOULD CONFIGURE YOUR ROOT PATH HERE **********
# **********        CHANGE THE PATH FOR home          **********
# ************************************************************
home = .
db_path = $home/Marway_RCM_Certs/RCM_CA_Masters

# Mandatory definitions for CA cert database management:
serial          = $db_path/serial
database        = $db_path/index.txt
new_certs_dir   = $db_path/Signed_Certs
certificate     = $db_path/rcm_ca_root_cert.pem
private_key     = $db_path/rcm_ca_root_key.pem

default_md      = sha256
default_days    = 3650

policy          = rcm_signing_policy
x509_extensions = rcm_signing_v3
copy_extensions = copy



[ rcm_signing_policy ]

# These fields are collectively known as the Distinguished Name (DN). The
# policy section defines which fields of the PDU's DN are required or
# optional, and which must exactly match the CA certificate's DN.

# "match" means the PDU CSR cert must match the CA cert.
# "supplied" means any value is OK, but not empty.
# "optional" means any value is OK, including empty.
# Any fields not mentioned are not included in the signed cert.

# Since we're trying to create a CA root for our own private use,
# it seems like a good idea that at least the country name and
# organization name should match. Additionally, it is probably
# useful to include a company contact in the email. However, you
# can edit these to enforce whatever you want.

countryName             = match
stateOrProvinceName     = optional
localityName            = optional
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = supplied
```

```
#-----------------------------------------------------------------------------
#
# REQ -- configs which apply to using the `openssl req` command.
#
#-----------------------------------------------------------------------------

[ req ]

default_bits       = 2048
default_md         = sha256
distinguished_name = rcm_req_dn
string_mask         = nombstr



[ rcm_req_dn ]

countryName                = Country Name (2 letter code)
countryName_min               = 2
countryName_max               = 2
stateOrProvinceName           = State or Province Name (full name)
localityName               = Locality Name (e.g. city)
organizationName           = Organization Name (e.g. company)
organizationalUnitName      = Organizational Unit Name (e.g. division)
commonName                 = Common Name (e.g. IP or FQDN)
commonName_max             = 64
emailAddress               = Email Address
emailAddress_max           = 64

# ************************************************************
# USRCFG: *****    YOU SHOULD SET THE VALUES HERE     **********
# ************************************************************
countryName_default             = US
stateOrProvinceName_default     =
localityName_default            =
organizationName_default        =
organizationalUnitName_default  =
emailAddress_default            =
```

```
#-------------------------------------------------------------------------------
#
# Extensions -- alternate x509v3 configs depending on which command is in use.
#
#-------------------------------------------------------------------------------

[ rcm_signing_v3 ]
# These are the extensions to use when signing CSR certs.
# Defined as the default up in [marway_rcm_ca],
# so does not need to be defined in the CLI command.

basicConstraints         = CA:false
subjectKeyIdentifier     = hash
authorityKeyIdentifier   = keyid,issuer
subjectAltName           = email:move

#-------------------------------------------------------------------------------
# THIS WILL HAVE TO BE CHANGED FOR EVERY ADDRESS YOU HAVE A CSR FOR.
# Or you can script something to make it more convenient to generate multiple
# signed certificates.
#
# For specific IP addresses, use this format:
# ************************************************************
# USRCFG: *****   CHANGE THIS VALUE FOR EVERY CERT    **********
# ************************************************************
subjectAltName = IP:192.168.0.0
#
# For specific IP addresses, use this format:
# subjectAltName = DNS.1 = pdu1.yourdomain.tld
#
#-------------------------------------------------------------------------------



[ rcm_csr_v3 ]
# These are the extensions to use when creating a new CSR for a server.
# Must be specified in the CLI command: `-extensions = rcm_csr_v3`

subjectAltName = email:move                 # Used for PKIX compliance



[ rcm_ca_v3 ]
# These are the extensions to use when signing the CA cert.
# Must be specified in the CLI command: `-extensions = rcm_ca_v3`

subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always,issuer:always
basicConstraints        = CA:true
subjectAltName          = email:move
```

Change Log

- r3 — September 23, 2019 (gw)
  - Removed “Confidential” labeling.
- r2 — July 1, 2019 (gw)
  - Typographic fixes, and title change to include versions 2.2 and later.
- r1 — August 2, 2018 (gw)
  - Original release.