wg TBD
Internet-Draft
Intended status: Informational
Expires: February 10, 2021

R. Moskowitz
HTT Consulting
H. Birkholz
Fraunhofer SIT
L. Xia
Huawei
M. Richardson
Sandelman

August 9, 2020

# Guide for building an ECC pki
# draft-moskowitz-ecdsa-pki-09

Abstract

This memo provides a guide for building a PKI (Public Key Infrastructure) using openSSL. All certificates in this guide are ECDSA, P-256, with SHA256 certificates. Along with common End Entity certificates, this guide provides instructions for creating IEEE 802.1AR iDevID Secure Device certificates.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 10, 2021.

Table of Contents

## 1. Introduction

The IETF has a plethora of security solutions targeted at IoT. Yet all too many IoT products are deployed with no or improperly configured security. In particular resource constrained IoT devices and non-IP IoT networks have not been well served in the IETF.

Additionally, more IETF (e.g. DOTS, NETCONF) efforts are requiring secure identities, but are vague on the nature of these identities other than to recommend use of X.509 digital certificates and perhaps TLS.

This effort provides the steps, using the openSSL application, to create such a PKI of ECDSA certificates. The goal is that any developer or tester can follow these steps, create the basic objects needed and establish the validity of the standard/program design. This guide can even be used to create a production PKi, though additional steps need to be taken. This could be very useful to a small vendor needing to include 802.1AR [IEEE.802.1AR_2009] iDevIDs in their product.

This guide was tested with openSSL 1.1.0f on Fedora 26 and creates PEM-based certificates. DER based certificates fails (see Section 9.4).

## 2. Terms and Definitions

### 2.1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

### 2.2. Notations

This section will contain notations

### 2.3. Definitions

There are no draft specific definitions at this time

## 3. The Basic PKI feature set

A basic pki has two levels of hierarchy: Root and Intermediate. The Root level has the greatest risk, and is the least used. It only signs the Intermediate level signing certificate. As such, once the Root level is created and signs the Intermediate level certificate it can be locked up. In fact, the Root level could exist completely on a mSD boot card for an ARM small computer like a RaspberryPi. A copy of this card came be made and securely stored in a different location.

The Root level contains the Root certificate private key, a database of all signed certificates, and the public certificate. It can also contain the Intermediate level public certificate and a Root level CRL.

The Intermediate level contains the Intermediate certificate private key, the public certificate, a database of all signed certificates, the certificate trust chain, and Intermediate level CRL. It can also contain the End Entity public certificates. The private key file needs to be keep securely. For example as with the Root level, a mSD image for an ARM computer could contain the complete Intermediate level. This image is kept offline. The End Entity CSR is copied to it, signed, and then the signed certificate and updated database are moved to the public image that lacks the private key.

For a simple test pki, all files can be kept on a single system that is managed by the tester.

End Entities create a key pair and a Certificate Signing Request (CSR). The private key is stored securely. The CSR is delivered to the Intermediate level which uses the CSR to create the End Entity certificate. This certificate, along with the trust chain back to the root, is then returned to the End Entity.

There is more to a pki, but this suffices for most development and testing needs.

## 4. Getting started and the Root level

This guide was developed on a Fedora 26 armv7hl system (Cubieboard2 SoC). It should work on most Linux and similar systems. All work was done in a terminal window with extensive "cutting and pasting" from a draft guide into the terminal window. Users of this guide may find different behaviors based on their system.

## 4.1. Setting up the Environment

The first step is to create the pki environment. Modify the variables to suit your needs.

<sourcecode> file "setup1.sh"

```
# edit directory here, or override
export cadir=${cadir-/root/ca}
export rootca=${cadir}/root
export cfgdir=${cfgdir-$cadir}
export intdir=${cadir}/intermediate
export int1ardir=${cadir}/inter_1ar
export format=pem
export default_crl_days=65

mkdir -p $cadir/certs
mkdir -p $rootca
(cd $rootca
mkdir -p certs crl csr newcerts private
chmod 700 private
touch index.txt index.txt.attr
if [ ! -f serial ]; then echo 00 >serial; fi
)

sn=8

# edit these to suit
countryName="/C=US"
stateOrProvinceName="/ST=MI"
localityName="/L=Oak Park"
organizationName="/O=HTT Consulting"
#organizationalUnitName="/OU="
organizationalUnitName=
commonName="/CN=Root CA"
DN=$countryName$stateOrProvinceName$localityName
DN=$DN$organizationName$organizationalUnitName$commonName

echo $DN
export subjectAltName=email:postmaster@htt-consult.com

export default_crl_days=2048
</sourcecode>
```

Where:

dir
: Directory for certificate files

cadir
: Directory for Root certificate files

Format
: File encoding: PEM or DER
: At this time only PEM works

sn
: Serial Number length in bytes
: For a public CA the range is 8 to 19

The Serial Number length for a public pki ranges from 8 to 19 bytes. The use of 19 rather than 20 is to accommodate the hex representation of the Serial Number. If it has a one in the high order bit, DER encoding rules will place a 0x00 in front.

The DN and SAN fields are examples. Change them to appropriate values. If you leave one blank, it will be left out of the Certificate. "OU" above is an example of an empty DN object.

Create the file, $dir/openssl-root.cnf from the contents in Appendix A.1.

## 4.2. Create the Root Certificate

Next are the openssl commands to create the Root certificate keypair, and the Root certificate. Included are commands to view the file contents.

```
<sourcecode> file "rootcert.sh"
# Create passworded keypair file

if [ ! -f $rootca/private/ca.key.$format ]; then
    echo GENERATING KEY
    openssl genpkey $pass -aes256 -algorithm ec\
            -pkeyopt ec_paramgen_curve:prime256v1\
            -outform $format -pkeyopt ec_param_enc:named_curve\
            -out $rootca/private/ca.key.$format
    chmod 400 $rootca/private/ca.key.$format
    openssl pkey $passin -inform $format -in $rootca/private/ca.key.$format\
            -text -noout
fi

# Create Self-signed Root Certificate file
# 7300 days = 20 years; Intermediate CA is 10 years.

echo GENERATING and SIGNING REQ
openssl req -config $cfgdir/openssl-root.cnf $passin \
     -set_serial 0x$(openssl rand -hex $sn)\
     -keyform $format -outform $format\
     -key $rootca/private/ca.key.$format -subj "$DN"\
     -new -x509 -days 7300 -sha256 -extensions v3_ca\
     -out $cadir/certs/ca.cert.$format

#

openssl x509 -inform $format -in $cadir/certs/ca.cert.$format\
     -text -noout
openssl x509 -purpose -inform $format\
     -in $cadir/certs/ca.cert.$format -inform $format
</sourcecode>
```

## 5. The Intermediate level

### 5.1. Setting up the Intermediate Certificate Environment

The next part is to create the Intermediate pki environment. Modify the variables to suit your needs. In particular, set the variables for CRL and/or OCSP support.

```
   <sourcecode> file "intermediate_setup.sh"

   export intdir=${intdir-$cadir/intermediate}
   mkdir -p $intdir

   (
   cd $intdir
   mkdir -p certs crl csr newcerts private
   chmod 700 private
   touch index.txt index.txt.attr
   if [ ! -f serial ]; then echo 00 >serial; fi
   )

   sn=8 # hex 8 is minimum, 19 is maximum
   echo 1000 > $intdir/crlnumber

   # cd $dir
   export crlDP=
   # For CRL support use uncomment these:
   #crl=intermediate.crl.pem
   #crlurl=www.htt-consult.com/pki/$crl
   #export crlDP="URI:http://$crlurl"
   export default_crl_days=30
   export ocspIAI=
   # For OCSP support use uncomment these:
   #ocspurl=ocsp.htt-consult.com
   #export ocspIAI="OCSP;URI:http://$ocspurl"

   commonName="/CN=Signing CA"
   DN=$countryName$stateOrProvinceName$localityName$organizationName
   DN=$DN$organizationalUnitName$commonName
   echo $DN

   </sourcecode>
```

Create the file, $dir/openssl-intermediate.cnf from the contents in Appendix A.2. Uncomment lines for crlDistributionPoints and authorityInfoAccess if using CRLs or OSCP repectfully.

## 5.2. Create the Intermediate Certificate

Here are the openssl commands to create the Intermediate certificate keypair, Intermediate certificate signed request (CSR), and the Intermediate certificate. Included are commands to view the file contents.

```
<sourcecode> file "intermediate_cert.sh"
# Create passworded keypair file
```

```
if [ ! -f $intdir/private/intermediate.key.$format ]; then
    echo GENERATING intermediate KEY
    openssl genpkey $pass -aes256 -algorithm ec \
            -pkeyopt ec_paramgen_curve:prime256v1 \
            -outform $format -pkeyopt ec_param_enc:named_curve\
            -out $intdir/private/intermediate.key.$format
    chmod 400 $intdir/private/intermediate.key.$format
    openssl pkey $passin -inform $format\
            -in $intdir/private/intermediate.key.$format -text -noout
fi

# Create the CSR

echo GENERATING and SIGNING REQ intermediate
openssl req -config $cfgdir/openssl-root.cnf $passin \
    -key $intdir/private/intermediate.key.$format -batch \
    -keyform $format -outform $format -subj "$DN" -new -sha256\
    -out $intdir/csr/intermediate.csr.$format
openssl req -text -noout -verify -inform $format\
    -in $intdir/csr/intermediate.csr.$format


# Create Intermediate Certificate file

openssl rand -hex $sn > $intdir/serial # hex 8 is minimum, 19 is maximum

if [ ! -f $cadir/certs/intermediate.cert.pem ]; then
    # Note 'openssl ca' does not support DER format
    openssl ca -config $cfgdir/openssl-root.cnf -days 3650 $passin \
            -extensions v3_intermediate_ca -notext -md sha256 -batch \
            -in $intdir/csr/intermediate.csr.$format\
            -out $cadir/certs/intermediate.cert.pem
    chmod 444 $cadir/certs/intermediate.cert.$format
    rm -f $cadir/certs/ca-chain.cert.$format
fi

openssl verify -CAfile $cadir/certs/ca.cert.$format\
     $cadir/certs/intermediate.cert.$format

openssl x509 -noout -text -in $cadir/certs/intermediate.cert.$format

# Create the certificate chain file

if [ ! -f $cadir/certs/ca-chain.cert.$format ]; then
    cat $cadir/certs/intermediate.cert.$format\
        $cadir/certs/ca.cert.$format > $cadir/certs/ca-chain.cert.$format
    chmod 444 $cadir/certs/ca-chain.cert.$format
fi
```

</sourcecode>

### 5.3. Create a Server EE Certificate

Here are the openssl commands to create a Server End Entity certificate keypair, Server certificate signed request (CSR), and the Server certificate. Included are commands to view the file contents.

<sourcecode> file "end-server.sh"

```
commonName=
DN=$countryName$stateOrProvinceName$localityName
DN=$DN$organizationName$organizationalUnitName$commonName
echo $DN
serverfqdn=www.example.com
emailaddr=postmaster@htt-consult.com
export subjectAltName="DNS:$serverfqdn, email:$emailaddr"
echo $subjectAltName
openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:prime256v1\
    -pkeyopt ec_param_enc:named_curve\
    -out $dir/private/$serverfqdn.key.$format
chmod 400 $dir/private/$serverfqdn.$format
openssl pkey -in $dir/private/$serverfqdn.key.$format -text -noout
openssl req -config $dir/openssl-intermediate.cnf\
    -key $dir/private/$serverfqdn.key.$format \
    -subj "$DN" -new -sha256 -out $dir/csr/$serverfqdn.csr.$format

openssl req -text -noout -verify -in $dir/csr/$serverfqdn.csr.$format

openssl rand -hex $sn > $dir/serial # hex 8 is minimum, 19 is maximum
# Note 'openssl ca' does not support DER format
openssl ca -config $dir/openssl-intermediate.cnf -days 375\
    -extensions server_cert -notext -md sha256 \
    -in $dir/csr/$serverfqdn.csr.$format\
    -out $dir/certs/$serverfqdn.cert.$format
chmod 444 $dir/certs/$serverfqdn.cert.$format

openssl verify -CAfile $dir/certs/ca-chain.cert.$format\
     $dir/certs/$serverfqdn.cert.$format
openssl x509 -noout -text -in $dir/certs/$serverfqdn.cert.$format
```

</sourcecode>

### 5.4. Create a Client EE Certificate

Here are the openssl commands to create a Client End Entity certificate keypair, Client certificate signed request (CSR), and the Client certificate. Included are commands to view the file contents.

```
   <sourcecode> file "end-client-dn.sh"
   commonName=
   UserID="/UID=rgm"
   DN=$countryName$stateOrProvinceName$localityName
   DN=$DN$organizationName$organizationalUnitName$commonName$UserID
   echo $DN
   clientemail=rgm@example.com
   </sourcecode>

<sourcecode> file "end-client.sh"
export subjectAltName="email:$clientemail"
echo $subjectAltName

if [ ! -f $intdir/private/$clientemail.key.$format ]; then
    openssl genpkey $pass -algorithm ec -pkeyopt ec_paramgen_curve:prime256v1\
            -pkeyopt ec_param_enc:named_curve\
            -out $intdir/private/$clientemail.key.$format
    chmod 400 $intdir/private/$clientemail.key.$format
    openssl pkey $passin -in $intdir/private/$clientemail.key.$format -text -
noout
fi

openssl req -config $cfgdir/openssl-intermediate.cnf $passin \
    -key $intdir/private/$clientemail.key.$format \
    -subj "$DN" -new -sha256 -out $intdir/csr/$clientemail.csr.$format

openssl req -text -noout -verify\
    -in $intdir/csr/$clientemail.csr.$format

openssl rand -hex $sn > $intdir/serial # hex 8 is minimum, 19 is maximum
# Note 'openssl ca' does not support DER format
openssl ca -config $cfgdir/openssl-intermediate.cnf -days 375\
    -extensions usr_cert -notext -md sha256 $passin \
    -in   $intdir/csr/$clientemail.csr.$format -batch\
    -out  $cadir/certs/$clientemail.cert.$format
chmod 444 $cadir/certs/$clientemail.cert.$format

openssl verify -CAfile $cadir/certs/ca-chain.cert.$format\
     $cadir/certs/$clientemail.cert.$format
openssl x509 -noout -text -in $cadir/certs/$clientemail.cert.$format
</sourcecode>
```

## 6. The 802.1AR Intermediate level

### 6.1. Setting up the 802.1AR Intermediate Certificate Environment

The next part is to create the 802.1AR Intermediate pki environment. This is very similar to the Intermediate pki environment. Modify the variables to suit your needs.

```
<sourcecode> file "intermediate_1ar_setup.sh"
export dir=$cadir/8021ARintermediate
mkdir $dir
cd $dir
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
sn=8 # hex 8 is minimum, 19 is maximum
echo 1000 > $dir/crlnumber

# cd $dir
export crlDP=
# For CRL support use uncomment these:
#crl=8021ARintermediate.crl.pem
#crlurl=www.htt-consult.com/pki/$crl
#export crlDP="URI:http://$crlurl"
export default_crl_days=30
export ocspIAI=
# For OCSP support use uncomment these:
#ocspurl=ocsp.htt-consult.com
#export ocspIAI="OCSP;URI:http://$ocspurl"

countryName="/C=US"
stateOrProvinceName="/ST=MI"
localityName="/L=Oak Park"
organizationName="/O=HTT Consulting"
organizationalUnitName="/OU=Devices"
#organizationalUnitName=
commonName="/CN=802.1AR CA"
DN=$countryName$stateOrProvinceName$localityName$organizationName
DN=$DN$organizationalUnitName$commonName
echo $DN
export subjectAltName=email:postmaster@htt-consult.com
echo $subjectAltName
</sourcecode>
```

Create the file, $dir/openssl-8021ARintermediate.cnf from the contents in Appendix A.3. Uncomment lines for crlDistributionPoints and authorityInfoAccess if using CRLs or OSCP repectfully.

## 6.2. Create the 802.1AR Intermediate Certificate

Here are the openssl commands to create the 802.1AR Intermediate certificate keypair, 802.1AR Intermediate certificate signed request (CSR), and the 802.1AR Intermediate certificate. Included are commands to view the file contents.

```
<sourcecode> file "intermediate_1ar_cert.sh"
# Create passworded keypair file

openssl genpkey -aes256 -algorithm ec\
    -pkeyopt ec_paramgen_curve:prime256v1 \
    -outform $format -pkeyopt ec_param_enc:named_curve\
    -out $dir/private/8021ARintermediate.key.$format
chmod 400 $dir/private/8021ARintermediate.key.$format
openssl pkey -inform $format\
    -in $dir/private/8021ARintermediate.key.$format -text -noout

# Create the CSR

openssl req -config $cadir/openssl-root.cnf\
    -key $dir/private/8021ARintermediate.key.$format \
    -keyform $format -outform $format -subj "$DN" -new -sha256\
    -out $dir/csr/8021ARintermediate.csr.$format
openssl req -text -noout -verify -inform $format\
    -in $dir/csr/8021ARintermediate.csr.$format


# Create 802.1AR Intermediate Certificate file
# The following does NOT work for DER

openssl rand -hex $sn > $dir/serial # hex 8 is minimum, 19 is maximum
# Note 'openssl ca' does not support DER format
openssl ca -config $cadir/openssl-root.cnf -days 3650\
    -extensions v3_intermediate_ca -notext -md sha256\
    -in $dir/csr/8021ARintermediate.csr.$format\
    -out $dir/certs/8021ARintermediate.cert.pem

chmod 444 $dir/certs/8021ARintermediate.cert.$format

openssl verify -CAfile $cadir/certs/ca.cert.$format\
     $dir/certs/8021ARintermediate.cert.$format

openssl x509 -noout -text\
     -in $dir/certs/8021ARintermediate.cert.$format

# Create the certificate chain file

cat $dir/certs/8021ARintermediate.cert.$format\
   $cadir/certs/ca.cert.$format > $dir/certs/ca-chain.cert.$format
chmod 444 $dir/certs/ca-chain.cert.$format

</sourcecode>
```

### 6.3. Create an 802.1AR iDevID Certificate

Here are the openssl commands to create a 802.1AR iDevID certificate keypair, iDevID certificate signed request (CSR), and the iDevID certificate. Included are commands to view the file contents.

<sourcecode> file "idevid-csr-cert.sh"

```
DevID=Wt1234
countryName=
stateOrProvinceName=
localityName=
organizationName="/O=HTT Consulting"
organizationalUnitName="/OU=Devices"
commonName=
serialNumber="/serialNumber=$DevID"
DN=$countryName$stateOrProvinceName$localityName
DN=$DN$organizationName$organizationalUnitName$commonName
DN=$DN$serialNumber
echo $DN

# hwType is OID for HTT Consulting, devices, sensor widgets
export hwType=1.3.6.1.4.1.6715.10.1
export hwSerialNum=01020304 # Some hex
export subjectAltName="otherName:1.3.6.1.5.5.7.8.4;SEQ:hmodname"
echo  $hwType - $hwSerialNum

if [ ! -f $dir/private/$DevID.key.$format ]; then
    openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:prime256v1\
            -pkeyopt ec_param_enc:named_curve\
            -out $dir/private/$DevID.key.$format
    chmod 400 $dir/private/$DevID.key.$format
fi

openssl pkey -in $dir/private/$DevID.key.$format -text -noout
openssl req -config $cfgdir/openssl-8021ARintermediate.cnf\
    -key $dir/private/$DevID.key.$format \
    -subj "$DN" -new -sha256 -out $dir/csr/$DevID.csr.$format

openssl req -text -noout -verify\
    -in $dir/csr/$DevID.csr.$format
openssl asn1parse -i -in $dir/csr/$DevID.csr.pem
# offset of start of hardwareModuleName and use that in place of 189
openssl asn1parse -i -strparse 189 -in $dir/csr/$DevID.csr.pem

openssl rand -hex $sn > $dir/serial # hex 8 is minimum, 19 is maximum
# Note 'openssl ca' does not support DER format
openssl ca -config $cfgdir/openssl-8021ARintermediate.cnf -days 375\
```

```
    -extensions 8021ar_idevid -notext -md sha256 \
    -in $dir/csr/$DevID.csr.$format\
    -out $dir/certs/$DevID.cert.$format
chmod 444 $dir/certs/$DevID.cert.$format

openssl verify -CAfile $dir/certs/ca-chain.cert.$format\
     $dir/certs/$DevID.cert.$format
openssl x509 -noout -text -in $dir/certs/$DevID.cert.$format
openssl asn1parse -i -in $dir/certs/$DevID.cert.pem

# offset of start of hardwareModuleName and use that in place of 493
openssl asn1parse -i -strparse 493 -in $dir/certs/$DevID.cert.pem

</sourcecode>
```

## 7. Setting up a CRL for an Intermediate CA

This part provides CRL support to an Intermediate CA. In this memo it applies to both Intermediate CAs. Set the crlDistributionPoints as provided via the environment variables.

### 7.1. Create (or recreate) the CRL

It is simple to create the CRL. The CRL consists of the certificates flagged with an R (Revoked) in index.txt:

```
<sourcecode> file "crl-creation.sh"

# Select which Intermediate level
intermediate=intermediate
#intermediate=8021ARintermediate
dir=$cadir/$intermediate
crl=$intermediate.crl.pem

# Create CRL file
openssl ca -config $dir/openssl-$intermediate.cnf \
      -gencrl -out $dir/crl/$crl
chmod 444 $dir/crl/$crl

openssl crl -in $dir/crl/$crl -noout -text

</sourcecode>
```

### 7.2. Revoke a Certificate

Revoking a certificate is a two step process. First identify the target certificate, examples are listed below. Revoke it then publish a new CRL.

```
<sourcecode> file "revoke-step1.sh"

targetcert=fqdn
#targetcert=clientemail
#targetcert=DevID

openssl ca -config $dir/openssl-$intermediate.cnf\
 -revoke $dir/certs/$targetcert.cert.$format

</sourcecode>
```

Recreate the CRL using Section 7.1.

## 8. Setting up OCSP for an Intermediate CA

This part provides OCSP support to an Intermediate CA. In this memo it applies to both Intermediate CAs. Set the authorityInfoAccess as provided via the environment variables.

### 8.1. Create the OCSP Certificate

OCSP needs a signing certificate. This certificate must be signed by the CA that signed the certificate being checked. The steps to create this certificate is the similar to a Server certificate for the CA:

```
<sourcecode> file "ocsp-setup.sh"

# Select which Intermediate level
intermediate=intermediate
#intermediate=8021ARintermediate
# Optionally, password encrypt key pair
encryptkey=
#encryptkey=-aes256

# Create the key pair in Intermediate level $intermediate
cd $dir
openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:prime256v1\
    $encryptkey -pkeyopt ec_param_enc:named_curve\
    -out $dir/private/$ocspurl.key.$format
chmod 400 $dir/private/$ocspurl.$format
openssl pkey -in $dir/private/$ocspurl.key.$format -text -noout

# Create CSR
commonName=
DN=$countryName$stateOrProvinceName$localityName
DN=$DN$organizationName$organizationalUnitName$commonName
echo $DN
emailaddr=postmaster@htt-consult.com
export subjectAltName="DNS:$ocspurl, email:$emailaddr"
echo $subjectAltName
openssl req -config $dir/openssl-$intermediate.cnf\
    -key $dir/private/$ocspurl.key.$format \
    -subj "$DN" -new -sha256 -out $dir/csr/$ocspurl.csr.$format

openssl req -text -noout -verify -in $dir/csr/$ocspurl.csr.$format

# Create Certificate

openssl rand -hex $sn > $dir/serial # hex 8 is minimum, 19 is maximum
# Note 'openssl ca' does not support DER format
openssl ca -config $dir/openssl-$intermediate.cnf -days 375\
    -extensions ocsp -notext -md sha256 \
    -in $dir/csr/$ocspurl.csr.$format\
    -out $dir/certs/$ocspurl.cert.$format
chmod 444 $dir/certs/$ocspurl.cert.$format

openssl verify -CAfile $dir/certs/ca-chain.cert.$format\
     $dir/certs/$ocspurl.cert.$format
openssl x509 -noout -text -in $dir/certs/$ocspurl.cert.$format

</sourcecode>
```

### 8.2. Revoke a Certificate

Revoke the certificate as in Section 7.2. The OCSP responder SHOULD detect the flag change in index.txt and, when queried respond appropriately.

### 8.3. Testing OCSP with Openssl

OpenSSL provides a simple OCSP service that can be used to test the OCSP certificate and revocation process (Note that this only reads the index.txt to get the certificate status at startup).

In a terminal window, set variables dir and ocspurl (examples below), then run the simple OCSP service:

<sourcecode> file "run-ocsp-server.sh"

```
dir=/root/ca/intermediate
ocspurl=ocsp.htt-consult.com

openssl ocsp -port 2560 -text -rmd sha256\
      -index $dir/index.txt \
      -CA $dir/certs/ca-chain.cert.pem \
      -rkey $dir/private/$ocspurl.key.pem \
      -rsigner $dir/certs/$ocspurl.cert.pem \
      -nrequest 1
```

</sourcecode>

In another window, test out a certificate status with:

<sourcecode> file "test-ocsp-server.sh"

```
targetcert=fqdn
#targetcert=clientemail
#targetcert=DevID

openssl ocsp -CAfile $dir/certs/ca-chain.cert.pem \
      -url http://127.0.0.1:2560 -resp_text -sha256\
      -issuer $dir/certs/$intermediate.cert.pem \
      -cert $dir/certs/$targetcert.cert.pem
```

</sourcecode>

Revoke the certificate, Section 7.2, restart the test Responder again as above, then check the certificate status.

## 9. Footnotes

Creating this document was a real education in the state of openSSL, X.509 certificate guidance, and just general level of certificate awareness. Here are a few short notes.

### 9.1. Certificate Serial Number

The certificate serial number's role is to provide yet another way to maintain uniqueness of certificates within a pki as well as a way to index them in a data store. It has taken on other roles, most notably as a defense.

The CABForum guideline for a public CA is for the serial number to be a random number at least 8 octets long and no longer than 20 bytes. By default, openssl makes self-signed certificates with 8 octet serial numbers. This guide uses openssl's RAND function to generate the random value and pipe it into the -set_serial option. This number MAY have the first bit as a ONE; the DER encoding rules prepend such numbers with 0x00. Thus the limit of '19' for the variable 'ns'.

A private CA need not follow the CABForum rules and can use anything number for the serial number. For example, the root CA (which has no security risks mitigated by using a random value) could use '1' as its serial number. Intermediate and End Entity certificate serial numbers can also be of any value if a strong hash, like SHA256 used here. A value of 4 for ns would provide a sufficient population so that a CA of 10,000 EE certificates will have only a 1.2% probability of a collision. For only 1,000 certificates the probability drops to 0.012%.

The following was proposed on the openssl-user list as an alternative to using the RAND function:

Keep k bits (k/8 octets) long serial numbers for all your certificates, chose a block cipher operating on blocks of k bits, and operate this block cipher in CTR mode, with a proper secret key and secret starting counter. That way, no collision detection is necessary, you'll be able to generate 2^(k/2) unique k bits longs serial numbers (in fact, you can generate 2^k unique serial numbers, but after 2^(k/2) you lose some security guarantees).

With 3DES, k=64, and with AES, k=128.

### 9.2. Some OpenSSL config file limitations

There is a bit of inconsistency in how different parts and fields in the config file are used. Environment variables can only be used as values. Some fields can have null values, others cannot. The lack of allowing null fields means a script cannot feed in an environment variable with value null. In such a case, the field has to be removed from the config file.

The expectation is each CA within a PKI has its own config file, customized to the certificates supported by that CA.

### 9.3. subjectAltName support, or lack thereof

There is no direct openssl command line option to provide a subjectAltName for a certificate. This is a serious limitation. Per RFC 2818 [RFC2818] SAN is the object for providing email addresses and DNS addresses (FQDN), yet the common practice has been to use the commonName object within the distinguishedName object. How much of this is due to the difficulty in creating certificates with a SAN?

Thus the only way to provide a SAN is through the config file. And there are two approaches. This document uses an environment variable to provide the SAN value into the config file. Another approach is to use piping as in:

```
<sourcecode> file "san-creation-pipe.sh"
openssl req -new -sha256 -key domain.key\
 -subj "/C=US/ST=CA/O=Acme, Inc./CN=foo.com" -reqexts SAN\
  -config <(cat /etc/ssl/openssl.cnf\
   <(printf "[SAN]\nsubjectAltName=DNS:foo.com,DNS:www.foo.com"))\
  -out domain.csr

</sourcecode>
```

### 9.4. DER support, or lack thereof

The long, hard-fought battle with openssl to create a full DER pki failed. The is no facility to create a DER certificate from a DER CSR. It just is not there in the 'openssl ca' command. Even the 'openssl x509 -req' command cannot do this for a simple certificate.

Further, there is no 'hack' for making a certificate chain as there is with PEM. With PEM a simple concatenation of the certificates create a usable certificate chain. For DER, some recommend using PKCS#7 [RFC2315], where others point out that this format is poorly support 'in the field', whereas PKCS#12 [RFC7292] works for them.

Finally, openssl does supports converting a PEM certificate to DER:

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

This should also work for the keypair. However, in a highly constrained device it may make more sense to just store the raw keypair in the device's very limited secure storage.

## 10. IANA Considerations

TBD. May be nothing for IANA.

## 11. Security Considerations

### 11.1. Adequate Randomness

Creating certificates takes a lot of random numbers. A good source of random numbers is critical. Studies [WeakKeys] have found excessive amount of certificates, all with the same keys due to bad randomness on the generating systems. The amount of entropy available for these random numbers can be tested. On Fedora/Centos and most Linux systems use:

```
cat /proc/sys/kernel/random/entropy_avail
```

If the value is low (below 1000) check your system's randomness source. Is rng-tools installed? Consider adding an entropy collection service like haveged from issihosts.com/haveged.

### 11.2. Key pair Theft

During the certificate creation, particularly during keypair generation, the files are vulnerable to theft. This can be mitigate using umask. Before using openssl, set umask:

```
restore_mask=$(umask -p)
umask 077
```

Afterwards, restore it with:

```
$restore_mask
```

or just close the shell that was used, and start a new one. (The -p option to umask is a bash-ism)

There is nothing in these recipes that requires super-user on the system creating the certificates. Provided that adequate randomness is available, a virtual machine or container is entirely appropriate. Containers tend to have better access to randomness than virtual machines.

The scripts and configuration files and in particular, private keys, may be kept offline on a USB key for instance, and loaded when needed.

The OCSP server needs to be online and available to all clients that will use the certificates. This may mean available on the Internet. A firewall can protect the OCSP server, and port-forwards and/or ACL rules can restrict access to just the OCSP port. OCSP artifacts are signed by a key designed for that purpose only so do not require that the associated CA key be available online.

Generating new CRLs, however, requires that the CA signing key be online, which is one of the reasons for creating an intermediate CA.

## 12. Acknowledgments

This work was jump started by the excellent RSA pki guide by Jamie Nguyen. The openssl-user mailing list, with its many supportive experts; in particular: Rich Salz, Jakob Bolm, Viktor Dukhovni, and Erwann Abalea, was of immense help as was the openssl man pages website.

Finally, "Professor Google" was always ready to point to answers to questions like: "openssl subjectAltName on the command line". And the Professor, it seems, never tires of answering even trivial questions.

## 13. References

### 13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

### 13.2. Informative References

[IEEE.802.1AR_2009]
IEEE, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", IEEE 802.1AR-2009, DOI 10.1109/ieeestd.2009.5367679, December 2009, <http://ieeexplore.ieee.org/servlet/opac?punumber=5367676>.

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <https://www.rfc-editor.org/info/rfc2315>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC7292] Moriarty, K., Ed., Nystrom, M., Parkinson, S., Rusch, A., and M. Scott, "PKCS #12: Personal Information Exchange Syntax v1.1", RFC 7292, DOI 10.17487/RFC7292, July 2014, <https://www.rfc-editor.org/info/rfc7292>.

[WeakKeys]
Heninger, N., Durumeric, Z., Wustrow, E., and J. Halderman, "Detection of Widespread Weak Keys in Network Devices", July 2011, <https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final228.pdf>.

# Appendix A. OpenSSL config files

## A.1. OpenSSL Root config file

The following is the openssl-root.cnf file contents

```
# OpenSSL root CA configuration file.
# Copy to `$dir/openssl.cnf`.

[ ca ]
# `man ca`
default_ca = CA_default

[ CA_default ]
# Directory and file locations.
dir               = $ENV::rootca
cadir             = $ENV::cadir
format            = $ENV::format

certs             = $dir/certs
crl_dir           = $dir/crl
```

```
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
serial            = $dir/serial
RANDFILE          = $dir/private/.rand

# The root key and root certificate.
private_key       = $dir/private/ca.key.$format
certificate       = $cadir/certs/ca.cert.$format

# For certificate revocation lists.
crlnumber         = $dir/crlnumber
crl               = $dir/crl/ca.crl.pem
crl_extensions    = crl_ext
default_crl_days  = 30

# SHA-1 is deprecated, so use SHA-2 instead.
default_md        = sha256

name_opt          = ca_default
cert_opt          = ca_default
default_days      = 375
preserve          = no
policy            = policy_strict
copy_extensions   = copy

[ policy_strict ]
# The root CA should only sign intermediate certificates that match.
# See the POLICY FORMAT section of `man ca`.
countryName             = optional
stateOrProvinceName     = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = optional

[ policy_loose ]
# Allow the intermediate CA to sign a more
#   diverse range of certificates.
# See the POLICY FORMAT section of the `ca` man page.
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = optional

[ req ]
# Options for the `req` tool (`man req`).
default_bits        = 2048
```

```
distinguished_name  = req_distinguished_name
string_mask         = utf8only
req_extensions      = req_ext

# SHA-1 is deprecated, so use SHA-2 instead.
default_md          = sha256

# Extension to add when the -x509 option is used.
x509_extensions     = v3_ca

[ req_distinguished_name ]
# See <https://en.wikipedia.org/wiki/Certificate_signing_request>.
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name

# Optionally, specify some defaults.
# countryName_default             = US
# stateOrProvinceName_default     = MI
# localityName_default            = Oak Park
# 0.organizationName_default      = HTT Consulting
# organizationalUnitName_default  =

[ req_ext ]
subjectAltName = $ENV::subjectAltName

[ v3_ca ]
# Extensions for a typical CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true
# keyUsage = critical, digitalSignature, cRLSign, keyCertSign
keyUsage = critical, cRLSign, keyCertSign
subjectAltName = $ENV::subjectAltName

[ v3_intermediate_ca ]
# Extensions for a typical intermediate CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true, pathlen:0
# keyUsage = critical, digitalSignature, cRLSign, keyCertSign
keyUsage = critical, cRLSign, keyCertSign

[ crl_ext ]
# Extension for CRLs (`man x509v3_config`).
```

```
authorityKeyIdentifier=keyid:always

[ ocsp ]
# Extension for OCSP signing certificates (`man ocsp`).
basicConstraints = CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = critical, digitalSignature
extendedKeyUsage = critical, OCSPSigning
</sourcecode>
```

### A.2. OpenSSL Intermediate config file

The following is the openssl-intermediate.cnf file contents.

Remove the crlDistributionPoints to drop CRL support and authorityInfoAccess to drop OCSP support.

```
# OpenSSL intermediate CA configuration file.
# Copy to `$dir/intermediate/openssl-intermediate.cnf`.

[ ca ]
# `man ca`
default_ca = CA_default

[ CA_default ]
# Directory and file locations.
dir               = $ENV::intdir
cadir             = $ENV::cadir
format            = $ENV::format

certs             = $dir/certs
crl_dir           = $dir/crl
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
serial            = $dir/serial
RANDFILE          = $dir/private/.rand

# The Intermediate key and Intermediate certificate.
private_key       = $dir/private/intermediate.key.$format
certificate       = $cadir/certs/intermediate.cert.$format

# For certificate revocation lists.
crlnumber         = $dir/crlnumber
crl               = $dir/crl/intermediate.crl.pem
crl_extensions    = crl_ext
default_crl_days  = $ENV::default_crl_days
```

```
   # SHA-1 is deprecated, so use SHA-2 instead.
   default_md        = sha256

   name_opt          = ca_default
   cert_opt          = ca_default
   default_days      = 375
   preserve          = no
   policy            = policy_loose
   copy_extensions   = copy

   [ policy_strict ]
   # The root CA should only sign intermediate certificates that match.
   # See the POLICY FORMAT section of `man ca`.
   countryName             = optional
   stateOrProvinceName     = optional
   organizationName        = optional
   organizationalUnitName  = optional
   commonName              = optional

   [ policy_loose ]
   # Allow the intermediate CA to sign a more
   #  diverse range of certificates.
   # See the POLICY FORMAT section of the `ca` man page.
   countryName             = optional
   stateOrProvinceName     = optional
   localityName            = optional
   organizationName        = optional
   organizationalUnitName  = optional
   commonName              = optional
   UID                     = optional

   [ req ]
   # Options for the `req` tool (`man req`).
   default_bits        = 2048
   distinguished_name  = req_distinguished_name
   string_mask         = utf8only
   req_extensions      = req_ext

   # SHA-1 is deprecated, so use SHA-2 instead.
   default_md          = sha256

   # Extension to add when the -x509 option is used.
   x509_extensions     = v3_ca

   [ req_distinguished_name ]
   # See <https://en.wikipedia.org/wiki/Certificate_signing_request>.
   countryName                     = Country Name (2 letter code)
   stateOrProvinceName             = State or Province Name
```

```
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
UID                             = User ID

# Optionally, specify some defaults.
# countryName_default             = US
# stateOrProvinceName_default     = MI
# localityName_default            = Oak Park
# 0.organizationName_default      = HTT Consulting
# organizationalUnitName_default  =

[ req_ext ]
subjectAltName = $ENV::subjectAltName

[ v3_ca ]
# Extensions for a typical CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true
# keyUsage = critical, digitalSignature, cRLSign, keyCertSign
keyUsage = critical, cRLSign, keyCertSign

[ v3_intermediate_ca ]
# Extensions for a typical intermediate CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true, pathlen:0
# keyUsage = critical, digitalSignature, cRLSign, keyCertSign
keyUsage = critical, cRLSign, keyCertSign

[ usr_cert ]
# Extensions for client certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = client, email
nsComment = "OpenSSL Generated Client Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = critical,nonRepudiation,digitalSignature,keyEncipherment
extendedKeyUsage = clientAuth, emailProtection
# uncomment the following if the ENV variables set
# crlDistributionPoints = $ENV::crlDP
# authorityInfoAccess = $ENV::ocspIAI

[ server_cert ]
# Extensions for server certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
```

```
nsCertType = server
nsComment = "OpenSSL Generated Server Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
# uncomment the following if the ENV variables set
# crlDistributionPoints = $ENV::crlDP
# authorityInfoAccess = $ENV::ocspIAI

[ crl_ext ]
# Extension for CRLs (`man x509v3_config`).
authorityKeyIdentifier=keyid:always

[ ocsp ]
# Extension for OCSP signing certificates (`man ocsp`).
basicConstraints = CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = critical, digitalSignature
extendedKeyUsage = critical, OCSPSigning
</sourcecode>
```

### A.3. OpenSSL 802.1AR Intermediate config file

The following is the openssl-8021ARintermediate.cnf file contents.

Remove the crlDistributionPoints to drop CRL support and authorityInfoAccess to drop OCSP support.

```
# OpenSSL 8021ARintermediate CA configuration file.
# Copy to `$dir/8021ARintermediate/openssl-8021ARintermediate.cnf`.

[ ca ]
# `man ca`
default_ca = CA_default

[ CA_default ]
# Directory and file locations.
# dir               = /root/ca/8021ARintermediate
dir               = $ENV::dir
cadir             = $ENV::cadir
format            = $ENV::format

certs             = $dir/certs
crl_dir           = $dir/crl
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
```

```
   serial            = $dir/serial
   RANDFILE          = $dir/private/.rand

   # The root key and root certificate.
   private_key       = $dir/private/8021ARintermediate.key.$format
   certificate       = $dir/certs/8021ARintermediate.cert.$format

   # For certificate revocation lists.
   crlnumber         = $dir/crlnumber
   crl               = $dir/crl/ca.crl.pem
   crl_extensions    = crl_ext
   default_crl_days  = $ENV::default_crl_days

   # SHA-1 is deprecated, so use SHA-2 instead.
   default_md        = sha256

   name_opt          = ca_default
   cert_opt          = ca_default
   default_enddate   = 99991231235959Z # per IEEE 802.1AR
   preserve          = no
   policy            = policy_loose
   copy_extensions   = copy

   [ policy_strict ]
   # The root CA should only sign 8021ARintermediate
   #   certificates that match.
   # See the POLICY FORMAT section of `man ca`.
   countryName             = match
   stateOrProvinceName     = match
   organizationName        = match
   organizationalUnitName  = optional
   commonName              = optional

   [ policy_loose ]
   # Allow the 8021ARintermediate CA to sign
   #   a more diverse range of certificates.
   # See the POLICY FORMAT section of the `ca` man page.
   countryName             = optional
   stateOrProvinceName     = optional
   localityName            = optional
   organizationName        = optional
   organizationalUnitName  = optional
   commonName              = optional
   serialNumber            = optional

   [ req ]
   # Options for the `req` tool (`man req`).
   default_bits        = 2048
```

```
distinguished_name  = req_distinguished_name
string_mask         = utf8only
req_extensions      = req_ext

# SHA-1 is deprecated, so use SHA-2 instead.
default_md          = sha256

# Extension to add when the -x509 option is used.
x509_extensions     = v3_ca

[ req_distinguished_name ]
# See <https://en.wikipedia.org/wiki/Certificate_signing_request>.
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
serialNumber                    = Device Serial Number

# Optionally, specify some defaults.
0.organizationName_default      = HTT Consulting
organizationalUnitName_default  = Devices

[ req_ext ]
subjectAltName = $ENV::subjectAltName

[ hmodname ]
hwType = OID:$ENV::hwType
hwSerialNum = FORMAT:HEX,OCT:$ENV::hwSerialNum

[ v3_ca ]
# Extensions for a typical CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ v3_8021ARintermediate_ca ]
# Extensions for a typical
#   8021ARintermediate CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true, pathlen:0
# keyUsage = critical, digitalSignature, cRLSign, keyCertSign
keyUsage = critical, cRLSign, keyCertSign

[ 8021ar_idevid ]
```

```
   # Extensions for IEEE 802.1AR iDevID
   #   certificates (`man x509v3_config`).
   basicConstraints = CA:FALSE
   authorityKeyIdentifier = keyid,issuer:always
   keyUsage = critical, digitalSignature, keyEncipherment
   # uncomment the following if the ENV variables set
   # crlDistributionPoints = $ENV::crlDP
   # authorityInfoAccess = $ENV::ocspIAI

   [ crl_ext ]
   # Extension for CRLs (`man x509v3_config`).
   authorityKeyIdentifier=keyid:always

   [ ocsp ]
   # Extension for OCSP signing certificates (`man ocsp`).
   basicConstraints = CA:FALSE
   subjectKeyIdentifier = hash
   authorityKeyIdentifier = keyid,issuer
   keyUsage = critical, digitalSignature
   extendedKeyUsage = critical, OCSPSigning
   </sourcecode>
```

## Authors' Addresses


Robert Moskowitz
HTT Consulting

Oak Park, MI  48237

Email: rgm@labs.htt-consult.com

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt  64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Liang Xia
Huawei
No. 101, Software Avenue, Yuhuatai District
Nanjing
China

Email: Frank.xialiang@huawei.com

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca
URI: http://www.sandelman.ca/