Workgroup: wg TBD
Internet-Draft: draft-moskowitz-eddsa-pki-06
Published: 10 July 2023
Intended Status: Informational
Expires: 11 January 2024
Authors: R. Moskowitz HTT Consulting
H. Birkholz Fraunhofer SIT
M. Richardson Sandelman

Guide for building an EdDSA PKI

Abstract

This memo provides a guide for building a PKI (Public Key Infrastructure) using openSSL. Certificates in this guide can be either ED25519 or ED448 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 11 January 2024.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

The IETF has adopted the Edwards Elliptic Curve and related algorithms. These algorithms hold out the promise of greater efficiency and better understood security risks. This efficiency

could make that critical difference to allow them to be used in some constrained IoT devices.

The IETF has a plethora of security solutions targeted at IoT. Yet all too many IoT products are tested and 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 ED25519 or ED448 certificates [RFC8032]. 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] that references [RFC4108] iDevIDs in their product (Note: EdDSA certificates are not supported in even the forthcoming 802.1AR-2018; this is for future work).

This guide was originally developed with openSSL 1.1.1; it is updated using openSSL 3.0.9 on Fedora 38 and creates PEM-based certificates. It closely follows [ecdsa-pki]. Current updates follow some lessons learned in developing [drip-dki]

Previous versions had multiple openssl config files. This version has a single config file to support all the openssl commands used herein.

2. Terms and Definitions

2.1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. Notations

This section will contain notations

2.3. Definitions

There are no draft specific definitions at this time

3. Comparing ECDSA and EdDSA certificates

There are two differences between ECDSA and EdDSA certificates that impact the use of openSSL. There are no options with EdDSA, and thus the pkeyopt variable is not used.

Likewise there are no hash options. The hashes used by EdDSA are preset and not selectable. As such, none of the hash options should be needed.

It should be noted here that ED25519 certificates can be ~100 bytes smaller than corresponding ECDSA certificates not using ECDSA point-compression. This size difference may be critical in some devices and communication technologies. ED448 certificates are similar in size with ECDSA p256 certificates yet with a stronger security claim.

4. 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 uSD boot card for an ARM small computer like a RaspberryPi. A copy of this card can be made and securely stored in a different location.

The Root level contains the Root certificate private key, a database of all root signed certificates, and the public root 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 its 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.

5. Getting started and the Root level

This guide was originally developed on a Fedora 29-beta armv7hl system (Cubieboard2 SoC). It should work on most Linux and similar systems that support openSSL 1.1.1. Current work has been on with openSSL 3.0.9 on Fedora 38. All work was done in a terminal window with extensive "cutting and pasting" from this draft guide into the terminal window. Users of this guide may find different behaviors based on their system.

5.1. Setting up the Environment

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

```
export dir=/root/ca
export cadir=/root/ca
export format=pem
export algorithm=ed25519 # or ed448
mkdir $dir
cd $dir
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
touch serial
sn=8

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
```

```
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

DN      : Distinguished Name
             Policy is LOOSE; Recommended at least
             providing Common Name
```

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.cnf from the contents in Appendix A.

5.2. 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.

Environment variables are used to provide considerable flexibility in the certificate contents. One change is to move from certificate life in days to validity notBefore and notAfter days. This is not supported in the standard self-sign root certificate creation, so a "through-away" self-signed certificate is created first which is used to sign a certificate with the same key and proper validity dates which is then the root certificate.

It is not uncommon to get warning messages in "openssl ca" commands on not including some value it expects. For example a warning of not having "default_days", as we are using actual validity dates. These can be ignored.

OpenSSL does not allow empty variables in the config file. So in the example below, certextkeyusage is empty. There are a few req_ext sections (e.g. req_ext_bkes, bke, bks, and bk) to cover the more common sets of extensions used. If a different set is needed, add it to the config file.

It is not uncommon to get warning messages in "openssl ca" commands on not including some value it expects. For example a warning of not having "default_days", as we are using actual validity dates. These can be ignored.

```
# Create passworded keypair file

export encryptkey=""
#export encryptkey="-aes256"   # use to password protect private key

openssl genpkey $encryptkey -algorithm $algorithm\
    -outform $format -out $dir/private/ca.key.$format
chmod 400 $dir/private/ca.key.$format
openssl pkey -inform $format -in $dir/private/ca.key.$format\
    -text -noout



# Create Self-signed Throw-away Certificate file

export signprv="ca"
export signcert="cabase"
export basicConstraints="critical, CA:true"
export certkeyusage="critical, keyCertSign"
export certextkeyusage=""
export hwType="place holder" # all ENV in config must
export hwSerialNum="place holder" #   be define
export startdate=20230801000000Z  # YYYYMMDDHHMMSSZ
export enddate=20430731000000Z  # YYYYMMDDHHMMSSZ

openssl req -config $dir/openssl.cnf\
     -set_serial 0x$(openssl rand -hex $sn)\
     -keyform $format -outform $format\
     -key $dir/private/ca.key.$format -subj "$DN"\
     -new -x509 -extensions v3_ca\
     -out $dir/certs/$signcert.cert.$format

openssl x509 -in $dir/certs/$signcert.cert.$format\
      -inform $format -text -noout
openssl x509 -purpose -inform $format\
     -in $dir/certs/$signcert.cert.$format -inform $format
```

```
# Create Self-signed Root Certificate file

openssl req -config $dir/openssl.cnf -reqexts req_ext_bk\
    -key $dir/private/ca.key.$format \
    -subj "$DN" -new -out $dir/csr/ca.csr.$format

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

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

openssl ca -config $dir/openssl.cnf\
    -extensions v3_ca -notext \
    -in $dir/csr/ca.csr.$format\
    -out $dir/certs/ca.cert.$format
chmod 444 $dir/certs/ca.cert.$format

openssl x509 -inform $format -in $dir/certs/ca.cert.$format\
     -text -noout
openssl x509 -purpose -inform $format\
     -in $dir/certs/ca.cert.$format -inform $format

openssl x509 -in $dir/certs/ca.cert.$format\
      -out $dir/certs/ca.cert.der -outform der
```

6. The Intermediate level

6.1. 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.

```
export dir=$cadir/intermediate
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 $dir
export crlDP=
# For CRL support use uncomment these:
#crl=intermediate.crl.pem
#crlurl=www.htt-consult.com/pki/$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"
export signprv="ca"
export signcert="ca"
export basicConstraints="critical, CA:true, pathlen:0"
export certkeyusage="critical, cRLSign, keyCertSign"
export certextkeyusage=""

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

> The Intermediate level CA now uses the same openssl config file as the root. Just copy it from the main directory into the Intermediate.

6.2. 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.

```
# Create passworded keypair file

export encryptkey=""
#export encryptkey="-aes256"   # use to password protect private key

openssl genpkey $encryptkey -algorithm $algorithm\
    -outform $format -out $dir/private/intermediate.key.$format
chmod 400 $dir/private/intermediate.key.$format
openssl pkey -inform $format\
    -in $dir/private/intermediate.key.$format -text -noout
```

```
# Create the CSR

openssl req -config $cadir/openssl.cnf -reqexts req_ext_bk\
    -key $dir/private/intermediate.key.$format \
    -keyform $format -outform $format -subj "$DN" -new\
    -out $dir/csr/intermediate.csr.$format
openssl req -text -noout -verify -inform $format\
    -in $dir/csr/intermediate.csr.$format



# Create Intermediate Certificate file

export startdate=230801000000Z  # YYMMDDHHMMSSZ
export enddate=20340731000000Z  # YYYYMMDDHHMMSSZ
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.cnf\
    -extensions v3_intermediate_ca -notext \
    -in $dir/csr/intermediate.csr.$format\
    -out $dir/certs/intermediate.cert.pem

chmod 444 $dir/certs/intermediate.cert.$format

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

openssl x509 -noout -text -in $dir/certs/intermediate.cert.$format
openssl x509 -in $dir/certs/intermediate.cert.$format\
      -out $dir/certs/intermediate.cert.der -outform der

# Create the certificate chain file

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

6.3. 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.

If EE certificates are created at a different time than the Intermediate signing certificate, care needs to be taken that all variables are properly reset.

```
export cadir=$cadir/intermediate
commonName="/CN=Web Services"
DN=$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
export signprv="intermediate"
export signcert="intermediate"
export basicConstraints="CA:FALSE"
export certkeyusage="critical, digitalSignature, keyEncipherment"
export certextkeyusage="serverAuth"
export encryptkey=""
#export encryptkey="-aes256"   # use to password protect private key

openssl genpkey $encryptkey -algorithm $algorithm\
    -out $dir/private/$serverfqdn.key.$format
chmod 400 $dir/private/$serverfqdn.key.$format
openssl pkey -in $dir/private/$serverfqdn.key.$format -text -noout
openssl req -config $cadir/openssl.cnf -reqexts req_ext_bkes\
    -key $dir/private/$serverfqdn.key.$format \
    -subj "$DN" -new -out $dir/csr/$serverfqdn.csr.$format

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



export startdate=230801000000Z  # YYMMDDHHMMSSZ
export enddate=240731000000Z  # YYMMDDHHMMSSZ
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.cnf\
    -extensions server_cert -notext \
    -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

openssl x509 -in $dir/certs/$serverfqdn.cert.$format\
      -out $dir/certs/$serverfqdn.cert.der -outform der
```

6.4. 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.

For a Client certificate with "no" subject and only a subjectAltName, set the variable DN to "/". Also the subjectAltName MUST be marked "critical".

```
commonName=
UserID="/UID=rgm"
DN=$countryName$stateOrProvinceName$localityName
DN=$DN$organizationName$organizationalUnitName$commonName$UserID
echo $DN
clientemail=rgm@example.com
export subjectAltName="email:$clientemail"
echo $subjectAltName
export basicConstraints="CA:FALSE"
export certkeyusage="critical, digitalSignature, keyEncipherment"
export certextkeyusage=""
export encryptkey=""
#export encryptkey="-aes256"   # use to password protect private key

openssl genpkey $encryptkey -algorithm $algorithm\
    -out $dir/private/$clientemail.key.$format
chmod 400 $dir/private/$clientemail.key.$format
openssl pkey -in $dir/private/$clientemail.key.$format -text -noout
openssl req -config $dir/openssl.cnf -reqexts req_ext_bks\
    -key $dir/private/$clientemail.key.$format \
    -subj "$DN" -new -out $dir/csr/$clientemail.csr.$format

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

```
export startdate=230801000000Z  # YYMMDDHHMMSSZ
export enddate=240731000000Z  # YYMMDDHHMMSSZ
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.cnf\
    -extensions usr_cert -notext \
    -in $dir/csr/$clientemail.csr.$format\
    -out $dir/certs/$clientemail.cert.$format
chmod 444 $dir/certs/$clientemail.cert.$format

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

openssl x509 -in $dir/certs/$clientemail.cert.$format\
      -out $dir/certs/$clientemail.cert.der -outform der
```

7. The 802.1AR Intermediate level

7.1. Setting up the 802.1AR Intermediate Certificate Environment

There is no longer a need for a special 802.1AR Intermediate Certificate CA. The regular Intermediate Certificate CA may be used for 802.1AR iDevID certificates. A special CA may be set up by following the steps outlined in Section 6, but into a special intermediate8021AR directory.

The difference with 802.1AR device certificates may be in including in the subject the device serial number. These certificates MUST have an afterDate of forever and a specific subjectAltName. Details for these follow.

7.2. 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.

```
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

openssl genpkey -algorithm $algorithm\
    -out $dir/private/$DevID.key.$format
chmod 400 $dir/private/$DevID.key.$format
openssl pkey -in $dir/private/$DevID.key.$format -text -noout
openssl req -config $dir/openssl.cnf -reqexts req_ext_8021AR \
    -key $dir/private/$DevID.key.$format \
    -subj "$DN" -new -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 169
openssl asn1parse -i -strparse 169 -in $dir/csr/$DevID.csr.pem
```

```
export startdate=230801000000Z  # YYMMDDHHMMSSZ
export enddate=99991231235959Z # per IEEE 802.1AR
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.cnf\
    -extensions 8021ar_idevid -notext \
    -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 367
openssl asn1parse -i -strparse 367 -in $dir/certs/$DevID.cert.pem

openssl x509 -in $dir/certs/$DevID.cert.$format\
      -out $dir/certs/$DevID.cert.der -outform der
```

8. 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.

8.1. 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:

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

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

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

8.2. 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.

```
targetcert=$serverfqdn
#targetcert=$clientemail
#targetcert=$DevID

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

Recreate the CRL using Section 8.1.

9. 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.

9.1. 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:

```
# 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 $algorithm\
    $encryptkey -out $dir/private/$ocspurl.key.$format
chmod 400 $dir/private/$ocspurl.key.$format
openssl pkey -in $dir/private/$ocspurl.key.$format -text -noout

# Create CSR
commonName="/CN=ocsp"
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.cnf -reqexts req_ext_bks \
    -key $dir/private/$ocspurl.key.$format \
    -subj "$DN" -new -out $dir/csr/$ocspurl.csr.$format

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



# Create Certificate

export startdate=230801000000Z  # YYMMDDHHMMSSZ
export enddate=99991231235959Z # per IEEE 802.1AR
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.cnf\
    -extensions ocsp -notext \
    -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
```

9.2. Revoke a Certificate

Revoke the certificate as in Section 8.2. The OCSP responder SHOULD detect the flag change in index.txt and, when queried respond appropriately.

9.3. 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:

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

openssl ocsp -port 2560 -text\
      -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
```

In another window, test out a certificate status with:

```
targetcert=$serverfqdn
#targetcert=$clientemail
#targetcert=$DevID

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

Revoke the certificate, Section 8.2, restart the test Responder again as above, then check the certificate status.

10. 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.

10.1. 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.

10.2. 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.

10.3. subjectAltName support now works

Older versions of openSSL had limitations in support for subjectAltName (SAN). This is no longer the case. This document sets up the SAN in the config file. Alternatively, the "-addext" option can be used directly in the command line.

10.4. Certificates with only subjectAltName

In RFC 5280 [RFC5280] (sec 4.2.1.6): if the only subject identity in the certificate is in subjectAltName, then Subject MUST be empty and subjectAltName MUST be marked as critical.

This can be achieved with the variable DN=/ and subjectAltName (example given):

```
DN=/
export subjectAltName=critical,email:postmaster@htt-consult.com
```

10.5. DER support, or lack thereof

The long, hard-fought battle with openssl to create a full DER PKI failed. There 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 support 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.

11. IANA Considerations

TBD. May be nothing for IANA.

12. Security Considerations

This section is a complete copy of [ecdsa-pki]. Changes will be made if anything is found specific to either ECDSA or EdDSA.

12.1. 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 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.

12.2. 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
```

13. Acknowledgments

This work is possible because of the availability of openSSL 1.1.1. As in [ecdsa-pki], the openssl-user mailing list, with its many supportive experts, was of immense help in the nuance differences between ECDSA and EdDSA.

14. References

14.1. 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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

14.2. Informative References

[drip-dki] Moskowitz, R. and S. W. Card, "The DRIP DET public Key Infrastructure", Work in Progress, Internet-Draft, draft-moskowitz-drip-dki-06, 27 June 2023, <https://datatracker.ietf.org/doc/html/draft-moskowitz-drip-dki-06>.

[ecdsa-pki] Moskowitz, R., Birkholz, H., Xia, L., and M. Richardson, "Guide for building an ECC pki", Work in Progress, Internet-Draft, draft-moskowitz-ecdsa-pki-10, 31 January 2021, <https://datatracker.ietf.org/doc/html/draft-moskowitz-ecdsa-pki-10>.

[IEEE 802.1AR] IEEE, "IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity", DOI 10.1109/ieeestd.2018.8423794, 31 July 2018, <http://dx.doi.org/10.1109/ieeestd.2018.8423794>.

[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>.

[RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, DOI 10.17487/RFC4108, August 2005, <https://www.rfc-editor.org/info/rfc4108>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[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>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/

RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[WeakKeys] Heninger, N.H., Durumeric, Z.D., Wustrow, E.W., and J.A.H. 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 file

The following is the openssl.cnf file contents

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

[ ca ]
# `man ca`
default_ca = CA_default

[ CA_default ]
# Directory and file locations.
dir               = $ENV::dir
cadir             = $ENV::cadir
format            = $ENV::format
signprv           = $ENV::signprv
signcert          = $ENV::signcert
certkeyusage      = $ENV::certkeyusage
certextkeyusage   = $ENV::certextkeyusage
basicConstraints  = $ENV::basicConstraints

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

# The signing key and signing certificate.
private_key       = $cadir/private/$signprv.key.$format
certificate       = $cadir/certs/$signcert.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_startdate = $ENV::startdate
default_enddate   = $ENV::enddate
preserve          = no
policy            = policy_loose
copy_extensions   = copy

[ 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
serialNumber            = optional


[ req ]
# Options for the `req` tool (`man req`).
distinguished_name  = req_distinguished_name
string_mask         = utf8only
#req_extensions      = req_ext
default_crl_days  = 30

# 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

[ req_ext ]

[ req_ext_bkes ]
basicConstraints = $ENV::basicConstraints
keyUsage = $ENV::certkeyusage
extendedKeyUsage = $ENV::certextkeyusage
subjectAltName = $ENV::subjectAltName

[ req_ext_bke ]
basicConstraints = $ENV::basicConstraints
keyUsage = $ENV::certkeyusage
extendedKeyUsage = $ENV::certextkeyusage

[ req_ext_bks ]
basicConstraints = $ENV::basicConstraints
keyUsage = $ENV::certkeyusage
subjectAltName = $ENV::subjectAltName
```

```
[ req_ext_bk ]
basicConstraints = $ENV::basicConstraints
keyUsage = $ENV::certkeyusage

[ req_ext_8021AR ]
basicConstraints = $ENV::basicConstraints
keyUsage = $ENV::certkeyusage
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
basicConstraints = $ENV::basicConstraints
keyUsage = $ENV::certkeyusage

[ v3_intermediate_ca ]
# Extensions for a typical intermediate CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always
# basicConstraints = $ENV::basicConstraints
# keyUsage = $ENV::certkeyusage
# subjectAltName = $ENV::subjectAltName

[ usr_cert ]
# Extensions for client certificates (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always
# uncomment the following if the ENV variables set
# crlDistributionPoints = $ENV::crlDP
# authorityInfoAccess = $ENV::ocspIAI

[ usr_req ]
# Extensions for client certificates (`man x509v3_config`).
subjectAltName = critical, $ENV::subjectAltName


[ server_cert ]
# Extensions for server certificates (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always
# uncomment the following if the ENV variables set
# crlDistributionPoints = $ENV::crlDP
# authorityInfoAccess = $ENV::ocspIAI
```

```
[ 8021ar_idevid ]
# Extensions for IEEE 802.1AR iDevID
#   certificates (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always
# 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`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always
```

Authors' Addresses

Robert Moskowitz
HTT Consulting
Oak Park

Email: rgm@labs.htt-consult.com

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Michael C. Richardson
Sandelman Software Works

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