White Paper

Cisco IOS EST Certificate Provisioning with the libEST CA Server

What You Will Learn

The recently created Enrollment over Secure Transport (EST) protocol aims to handle authentication and certificate provisioning in a more robust manner. It offers a variety of advantages over its predecessor, the Simple Certificate Enrollment Protocol, or SCEP. Because of its advantages, Cisco IOS® and Cisco IOS XE products offer EST client functionality for secure certificate enrollment with EST-enabled certificate authorities (CAs). This document presents sample configurations of how you can use Cisco IOS Software to enroll RSA and ECC certificates with the open-source libEST's CA.

Contents

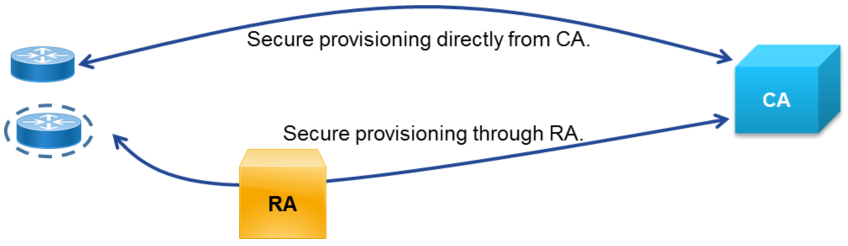
Introduction

Certificates have been used for authentication for a long time. The 802.1AR standard of the Institute of Electrical and Electronics Engineers (IEEE) uses them to prove the identity of a device. They are also widely used in Transport Layer Security (TLS), VPNs, and many other areas that require authentication. The certificates are usually generated by a trusted entity or certificate authority (CA), and they can be validated using a PKI root of trust and a certificate chain. Thus, the certificate provisioning mechanism needs to be secure and flexible. Figure 1 shows a high-level architecture of such a mechanism. A registration authority (RA) serves as an intermediary, authenticating the client before getting a certificate for him from the CA.

The most recently defined protocol that provides certificate provisioning is Enrollment over Secure Transport, the Internet Engineering Task Force's RFC 7030. EST profiles certificate enrollment for clients using Certificate Management over Cryptographic Message Syntax over a secure transport. According to the IETF, EST "describes a simple, yet functional, certificate management protocol targeting Public Key Infrastructure (PKI) clients that need to acquire client certificates and associated Certification Authority (CA) certificates. It also supports client-generated public/private key pairs as well as key pairs generated by the CA."

EST was a standardization effort that went through several iterations through the IETF. Multiple vendors and independent parties in the standards community participated in the effort. EST has many advantages over its SCEP predecessor. Support of elliptic curve cryptography (ECC), cryptographic algorithm agility, better authentication with TLS, and simplicity are some of them. To prove the protocol's value, Cisco itself has open-sourced libEST, an EST library that offers client and server functionality, in order to promote its adoption and interoperability across vendors. For more information on EST's advantages over other protocols, readers should refer to the PKI: Simplify Certificate Provisioning with EST whitepaper.

Figure 1: Common PKI Architecture with a Cisco Router or Virtual Router as the Client


Secure provisioning directly from CA.
CA
Secure provisioning through RA.
RA

Starting with releases 15.5(1)T and 3.14S, respectively, Cisco IOS and Cisco IOS XE Software includes EST client functionality for EST enrollment with CAs or RAs that support EST.

In the following sections we show you how to set up and enroll a Cisco IOS product with the EST server that is available in the open-source libEST library. We hope that this document will make it easier for adopters to move to a more secure certificate enrollment protocol.

How to Enroll RSA and ECC Certificates

In our examples, we used a Cisco IOS router running 15.5(3)M1 and libEST 1.1.1. Any later Cisco IOS or IOS XE release that supports EST should work as well. libEST was running on Ubuntu 14.04.3 with IP address 10.0.1.88. Any Linux distribution would also work. We assume that libEST and its dependencies (OpenSSL) are preinstalled in Ubuntu, and we do not go into details about installations in this document. For more information on installing libEST, refer to the libEST client use whitepaper. The Cisco IOS router uses EST to enroll two locally generated key pairs, one RSA and one ECC.

1. (Optional) Generate a TLS client certificate with OpenSSL

If the Cisco IOS release has a client certificate trust point that can be used for TLS client authentication, you can skip this step. The Cisco IOS command **`show crypto pki trustpoints`** can help identify trust points that can be used to authenticate to the server.

EST runs over TLS. The protocol needs to authenticate the client before allowing an enrollment request. Cisco® routers can use any certificate to authenticate themselves with the EST server as long as their root CA certificate is trusted at the EST server. This includes secure unique device identifier (SUDI) or other client certificates.

Cisco has been adding a SUDI certificate (IEEE's 802.1AR LDevID certificate equivalent) in its devices. A SUDI is a manufacturer's certificate and can be used to attest the authenticity of a device. More details about the Cisco IOS SUDI certificate profile can be seen in the appendix of the ACT2 SUDI documentation. The Cisco root CAs that sign the subCAs issuing the Cisco SUDI certificates are Cisco Root CA 2048 (crca2048), Cisco Root CA M2 (crcam2), and Cisco ECC Root CA documented in the Cisco PKI. Here is a sample output from a Cisco 4451-X Integrated Services Router's SUDI certificate:

```
Router#show crypto pki certificates
Certificate
  Status: Available
  Certificate Serial Number (hex): 000000
  Certificate Usage: General Purpose
  Issuer:
    cn=ACT2 SUDI CA
    o=Cisco
  Subject:
    Name: ISR4451-X/K9
    Serial Number: PID:ISR4451-X/K9 SN: XXXXXXXXXXX
    cn=ISR4451-X/K9
    ou=ACT-2 Lite SUDI
    o=Cisco
    serialNumber=PID:ISR4451-X/K9 SN:XXXXXXXXXXX
  Validity Date:
    start date: 22:51:47 UTC Dec 21 2013
    end   date: 22:51:47 UTC Dec 21 2023
  Associated Trustpoints: CISCO_IDEVID_SUDI
```

For our examples we are using privately generated RSA certificates because we have control of the libEST CA and we can import the client certificate root CA in the server's trust store so that it can authenticate the TLS client. Any other RSA certificate (SUDI or pre-existing) in the router will do, as long as we have the root CA available to import in libEST server's certificate trust store. In a real-world scenario a SUDI certificate that comes preinstalled with the router would be more practical. We are using a certificate we generate here only for demonstration purposes. Generating ECDSA certificates for TLS client authentication in IOS is very similar with the exception that the key and certificate request (CSR) is generated on the router and the certificate created externally. ECDSA specific details are addressed in Sections 1.3 and 3.1.

We will generate the client certificate in the EST server, but any Linux OS with OpenSSL installed would suffice.

1.1 Prepare to sign the client CSR

Create a filename **rootca_req.conf** for the client certificate root CA certificate extensions that includes

```
  [req]
  distinguished_name      = rootca_dn

  [rootca_dn]
```

```
organizationName         = "O="
commonName               = "CN="

[rootca_reqext]
keyUsage                 = critical,digitalSignature,keyCertSign,cRLSign
```

Generate the RSA key, the certificate request (filling in the organization and common name for the CA), and the certificate. That certificate is going to be the root CA certificate that will sign the client's certificate.

```
openssl genrsa -out rootca.key 2048
openssl req -new -x509 -sha256 -days 3650 -extensions rootca_reqext -config
rootca_req.conf -key rootca.key -out rootca.crt
```

Then create the extensions profile **`tls_client_req.conf`** for the client certificate that includes

```
[req]
distinguished_name       = tls_client_dn
# The extensions to add to a certificate request
x509_extensions          = tls_client_reqex

[tls_client_dn]
O                        = "O="
OU                       = "OU="
CN                       = "CN="
SN                       = "Serial="

[tls_client_reqex]
keyUsage = digitalSignature, keyEncipherment, dataEncipherment
```

1.2 Create the client CSR to be signed

Then generate the client RSA key, the certificate request (filling in the organization, organization unit, CN name, and serial number for the router):

```
openssl genrsa -out tls_client.key 2048
openssl req -new -x509 -sha256 -days 365 -config tls_client_req.conf -key
tls_client.key -out tls_client.csr
```

1.3 Generate the client certificate

Then generate the client certificate and sign it with the root CA key from above:

```
echo 01 > rootca.srl
openssl x509 -req -sha256 -days 365 -extensions tls_client_reqex -extfile
tls_client_req.conf -in tls_client.csr -CA rootca.crt -CAkey rootca.key -out
tls_client.crt
openssl x509 -outform pem -in tls_client.crt -out tls_client.pem
```

This newly created certificate and key pair will be imported in the router in order to authenticate itself to the EST server. The router requires the private key to have a passphrase. Let's use the passphrase "luckydog" and encrypt the private key:

```
openssl rsa -des3 -inform PEM -outform PEM -in tls_client.key -passout pass:luckydog
-out tls_client2.key
```

We have now generated the RSA certificate that the router will use to authenticate itself to the EST server.

Note on ECDSA certificates: Specifically for ECDSA client certificates, they are supported in IOS 15.6(1)T1 and later and IOS XE 3.16.2S and later. At the time of this writing (April 2016), externally generated keys and ECDSA certificates are not officially supported in IOS. Thus generating an ECDSA certificate that will be imported in the router in order to be authenticated by the server will be a very similar process without having to manually create the client CSR in Section 1.2. The CSR needs to be generated on the router as shown in Section 3.1. Then the CSR can be put in a file which will be used to generate the certificate as explained in Section 1.3.

2. Set up libEST CA

We now want to configure and run the example EST server available in libEST. We are assuming that the libEST and its dependencies (OpenSSL) are preinstalled in the Linux VM.

2.1 (Optional) Allow TLS 1.0 in the EST server. (Do this only for the Cisco IOS client in IOS and IOS XE versions before 15.6(1)T1 and 3.16.2S respectively.)

Even though EST mandates TLS 1.1 or later, the Cisco IOS TLS client might be using TLS 1.0 by default in older Cisco IOS releases (before 15.6(1)T1 and 3.16.2S). So we need allow TLS 1.0 in the libEST 1.1.1 server. **Enabling TLS 1.0 in the EST server is NOT recommended and should NOT take place if the router is running IOS 15.6(1)T1 and later or IOS XE 3.16.2S and later.** It violates RFC 7030 and introduces security vulnerabilities. We are only allowing it here to be able to interoperate with old Cisco IOS releases that did not support TLS 1.1 and later. In the EST server example code file **`example/server/estserver.c`**, add the following line in the **`main`** function while setting the est context around line 963:

```
    est_server_enable_tls10(ectx);
```

Alternatively, look for an estserver flag that enables TLS 1.0 (in newer libEST versions). If there is no estserver flag to enable TLS 1.0 and the **`est_server_enable_tls10`** is not defined, it can be defined in **`src/est/est_server.c`** as

```
EST_ERROR est_server_enable_tls10 (EST_CTX *ctx)
{
    if (!ctx) {
        EST_LOG_ERR("Null context");
        return (EST_ERR_NO_CTX);
    }
    ctx->enable_tls10 = 1;
    return (EST_ERR_NONE);
}
```

The function definition **`EST_ERROR est_server_enable_tls10(EST_CTX *ctx);`** needs to be added in **`src/est/est.h`**, and the variable **`enable_tls10`** needs to be defined in the **`struct est_ctx`** in **`src/est/est_locl.h`**. Finally, code

```
if (ectx->enable_tls10) {
        EST_LOG_INFO("Enabling TLS 1.0 support, not compliant with RFC 7030");
        SSL_CTX_set_options(ssl_ctx, SSL_OP_NO_SSLv2 |
                                    SSL_OP_NO_SSLv3 |
                                    SSL_OP_SINGLE_ECDH_USE |
                                    SSL_OP_NO_TICKET);
    } else {
        SSL_CTX_set_options(ssl_ctx, SSL_OP_NO_SSLv2 |
                                    SSL_OP_NO_SSLv3 |
                                    SSL_OP_NO_TLSv1 |
                                    SSL_OP_SINGLE_ECDH_USE |
                                    SSL_OP_NO_TICKET);
    }
```

needs to replace the **`SSL_CTX_set_options`** call around line 1336 in **`src/est/est_server_http.c`**.

Then compile and install libEST (**`make && make install`**). Now the EST server should be allowing TLS 1.0.

If you skipped the step above and in testing notice TLS error messages from the EST server, such as **`OSSL error: 140074427623168:error:1408A0C1:SSL routines:ssl3_get_client_hello:no shared cipher:s3_srvr.c:1411:`**, then is a result of the Cisco IOS Software using TLS 1.0, which the server is not permitting.

2.2 Create the CAs

Now let's set up the CAs. We want to test both RSA and ECC certificate enrollment, so we will use the libEST's extCA as the ECC CA, and the estCA as the RSA CA. Readers should note that these are not hardened production CAs. They are just example EST CAs that can be used for testing.

First, edit **`example/server/ESTcommon.sh`**

```
[ …Text omitted … ]
  EST_OPENSSLCMD_EXTCAECPARAMSFILE=$EST_OPENSSL_EXTCADIR/prime256v1.pem
  # if you want to use EC certificates set the ..._NEWKEY_PARAM like this:
  EST_OPENSSLCMD_EXTCANEWKEY_PARAM="-newkey  ec:$EST_OPENSSLCMD_EXTCAECPARAMSFILE"  # Make sure this line is NOT commented out
  # EST_OPENSSLCMD_EXTCANEWKEY_PARAM=" " # Make sure this line is commented out
 [ …Text omitted … ]
  #  EST_OPENSSLCMD_CANEWKEY_PARAM="-newkey  ec:$EST_OPENSSLCMD_CAECPARAMSFILE"  # Make sure this line is commented out
  EST_OPENSSLCMD_CANEWKEY_PARAM=" " # Make sure this line is NOT commented out
[ …Text omitted … ]
```

Then create the CAs:

```
./createCA
```

We will have two CAs, so we will run them on different ports. Let's say the RSA CA (estCA) will run on port 8085, and the ECC CA (extCA) on port 8086. Edit the **`example/server/runserverrsa.sh`** file and replace the **`./estserver...`** line with

```
./estserver -c estCA/private/estservercertandkey.pem -k
estCA/private/estservercertandkey.pem -r estrealm -v -p 8085 -n
```

Create a new file **example/server/runserverecc.sh** file that includes

```
export EST_TRUSTED_CERTS=./trustedcerts.crt
export EST_CACERTS_RESP=./extCA/cacert.crt
export EST_OPENSSL_CACONFIG=./extExampleCA.cnf
./estserver -c estCA/private/estservercertandkey.pem -k
estCA/private/estservercertandkey.pem -r estrealm -v -p 8086 -o
```

(Note that **-n** or **-o** options would suffice for running the servers.)

We now need to import the TLS client certificate root in our trust store so that the server can authenticate the TLS client certificate. Get the certificate from the previously created **rootca.crt** and put it in the **example/server/trustedcerts.crt**. In real-world case, a SUDI or any other certificate would be used in the router. The corresponding root CA certificate would need to go in the **example/server/trustedcerts.crt** trust store. For SUDI, the root certificates are Cisco Root CA 2048 (crca2048), Cisco Root CA M2 (crcam2), and Cisco ECC Root CA documented in the Cisco PKI.

2.3 Start the CAs

Then we can start the RSA and ECC EST CAs in the background. Readers should note that these are not hardened production CAs. They are just example EST CAs that can be used for testing.

Set the **LD_LIBRARY_PATH** environment variable and start the CAs:

```
export LD_LIBRARY_PATH=/usr/local/openssl/lib:/usr/local/est/lib
./runserverrsa.sh & ./runserverecc.sh &
```

To check the processes and potentially stop the EST servers from running, you could use

```
ps -ef | grep runserver
ps -ef | grep estserver
```

2.4 (Optional) Test the CAs with libEST's client to make sure they work

Now, optionally, we can use libEST's client to make sure that the CAs on the local machine are operational. Let's set the environment variables:

```
export LD_LIBRARY_PATH=/usr/local/openssl/lib:/usr/local/est/lib
export EST_OPENSSL_CACERT=./cacerts
```

The cacerts file is the one that is in the **example/server** directory.

In libEST's directory, let's create two directories to store the enrolled certificates and run the client:

```
mkdir testrsa
mkdir testecc
./example/client/estclient -v -e -s 127.0.0.1 -p 8085 -u estuser -h estpwd -o ./testrsa
./example/client/estclient -v -e -s 127.0.0.1 -p 8086 -u estuser -h estpwd -o ./testecc
```

The enrolled certificates should now be stored in the test directories. You can export the public key that is stored in these directories in pkcs7 format using

```
openssl base64 -d -in cert-0-0.pkcs7 | openssl pkcs7 -inform DER -outform PEM -print_certs -out cert.pem
```

If you want to verify that a private and a public key (in the certificate) in the test directories are from the same key pair, you can compare the output of

```
openssl rsa -noout -modulus -in key-x-x.pem
openssl x509 -noout -modulus -in cert.pem
```

For more information on how to use the libEST client and its use, refer to the libEST client use whitepaper.

3. Configure the router

After creating and starting the two CAs, we are ready to set up the routers for EST enrollment.

3.1 (If needed) Import the TLS client certificate from Section 1 into the router

Let's import the TLS RSA certificate that we created above. (If a SUDI or other certificate on the router is available, you can skip this step.)

```
crypto pki trustpoint LDevID
    enrollment terminal pem
    crl optional
    exit
crypto pki import LDevID pem terminal pass luckydog
```

At this point the router will ask to import the (intermediate) CA certificate, the private key, and the client certificate. For that we will use the `rootca.crt`, `tls_client2.key` and `tls_client.pem` we created above.

Note on ECDSA certificates: Specifically for ECDSA certificates, they are supported in IOS 15.6(1)T1 and later and IOS XE 3.16.2S and later. At the time of this writing (April 2016), externally generated keys and ECDSA (as we did above with RSA above) certificates are not officially supported. To import a PKI trust point for an ECDSA certificate, the ECC key should have previously been generated on the router and a CSR be manually exported as shown below

```
crypto key generate ec keysize 384 label ecdsakey
crypto pki trustpoint LDevID
    eckeypair ecdsakey
    enrollment terminal pem
    ! --- More certificate options here like subject-name, fqdn etc
    exit
crypto pki enroll LDevID
[ …Text omitted … ]
Certificate Request follows:

-----BEGIN CERTIFICATE REQUEST-----
MIIB+jCCAWMCAQAwgZgxGzAZBgNVBAM [ …Text omitted … ]
-----END CERTIFICATE REQUEST-----
```

```
---End - This line not part of the certificate request---

Redisplay enrollment request? [yes/no]: no
```

After the CSR printed out on the terminal is signed at the CA and the ECDSA certificate is generated (see Section 1), you can import the intermediate CA certificate and the router TLS ECDSA certificate using

```
crypto ca authenticate LDevID
crypto pki import LDevID
```

3.2 Import the TLS / EST server certificate root CA into the router

Now we need to import the root CA of the TLS certificate that the EST server will use, so that the router can authenticate the server:

```
crypto pki trustpoint TLS-root
  enrollment terminal PEM
  crl optional
  exit
crypto ca authenticate TLS-root
```

At this point the router will ask for the certificate. It can be retrieved from the EST server in `libEST/example/server/estCA/cacert.crt` or `libEST/example/server/extCA/cacert.crt`.

3.3 Configure EST client on the router

Now we can configure the EST client on the router. First let's create the RSA and ECC key pairs that we will get signed:

```
crypto key generate rsa modulus 2048 label rsakey
crypto key generate ec keysize 256 label ecckey
```

Then we will create the enrollment profiles for the RSA and ECC CAs:

```
crypto pki profile enrollment testrsa-prof
  method-est
  enrollment url https://10.0.1.88:8085
  enrollment credential LDevID
crypto pki profile enrollment testecc-prof
  method-est
  enrollment url https://10.0.1.88:8086
  enrollment credential LDevID
```

Then let's create the RSA and ECC trust points:

```
crypto pki trustpoint testrsa-ca
  usage ike
  subject-name CN=testrouter
  enrollment profile testrsa-prof
  revocation-check none
  rsakeypair rsakey
crypto pki trustpoint testecc-ca
  usage ike
  subject-name CN=testrouter
  enrollment profile testecc-prof
  revocation-check none
  eckeypair ecckey
```

The router is ready to enroll its certificate through the test CAs.

3.4 (Optional) Enable HTTP Basic Authentication

On top of TLS certificate client authentication, EST offers HTTP Basic and Digest Authentication. Cisco IOS Software can support HTTP Basic Authentication that is performed after the certificate client authentication. Certificate-less client authentication is not currently supported. To allow for HTTP Basic Authentication in Cisco IOS Software, use this command:

```
ip http client connection forceclose
```

The enrollment profiles (**`testrsa-prof`**, **`testecc-prof`**) used above also should contain the authentication URL and the username (estuser) and password (estpwd). For example:

```
crypto pki profile enrollment testrsa-prof
  method-est
  authentication url  https://10.0.1.88:8085
  enrollment url  https://estuser:estpwd@10.0.1.88:8085
  enrollment credential LDevID
```

Alternatively, the enrollment URL could remain **`https://10.0.1.88:8085`**, but then an HTTP client username and password would need to be configured on the router:

```
ip http client username estuser
ip http client password estpwd
```

The libEST server by default supports TLS certificate client authentication and HTTP Basic and Digest Authentication. So, finally the **`-o`** or **`-n`** option configured in scripts **`runserverrsa.sh`** and **`runserverecc.sh`** in Section 2.2 need be removed and the server scripts restarted.

4. Test IOS EST client with the CAs

To complete the enrollment, first authenticate and then enroll the trust points in the router:

```
crypto pki authenticate testrsa-ca
crypto pki enroll testrsa-ca
crypto pki authenticate testecc-ca
crypto pki enroll testecc-ca
```

At this point the router should have received two certificates from the CA. To check the certificates provisioned you can use the following commands:

```
show crypto pki certificate verbose testrsa-ca
show crypto pki certificate verbose testecc-ca
```

In the event that an enrollment fails, the administrator can stop an enrollment and clear the certificates to reauthenticate and enroll using

```
no crypto pki enroll testrsa-ca
no crypto pki certificate chain testrsa-ca
```

For debugging purposes on the router, there are a number of available debugs, such as

```
debug crypto pki transactions
debug crypto pki messages
debug crypto pki validation
debug crypto pki callbacks
```

After completing all the above steps, you should have two certificates provisioned on the routers. One of them should be RSA, and the other one ECC. These certificates can be used for authentication of various services like IKE (VPN) or HTTPS.

Conclusion

EST is a more secure and flexible certificate provisioning protocol with many advantages over its predecessor, SCEP. EST was standardized in 2013. Cisco IOS Software and Cisco IOS XE already support EST. In this document we saw how to set it up to enroll with libEST's EST server. Cisco IOS Software can be used with any CA or RA that supports EST. Because of its advantages, we expect EST's adoption among CAs and products to increase in various technology verticals.

References

EST RFC 7030: http://tools.ietf.org/html/rfc7030

libEST: https://github.com/cisco/libest

Cisco IOS EST client support:
http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_conn_pki/configuration/15-mt/sec-pki-15-mt-book/sec-est-client-supp-pki.html

Cisco IOS XE EST client support:
http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_conn_pki/configuration/xe-3s/sec-pki-xe-3s-book/sec-est-client-supp-pki.html

Acknowledgments

Panos Kampanakis (panosk[at]cisco[dot]com)
Technical Marketing Engineer

Thanks to Peter Panburana and Mike Luken for their valuable feedback.