

Open-Source EST Clients: How to Use Them for Secure Certificate Provisioning

What You Will Learn

The concept of a public key infrastructure (PKI) has existed for a long time. The PKI authenticates the identity of users and devices by using signed public key pairs in the form of certificates. The recently created Enrollment over Secure Transport (EST) protocol aims to provision certificates in a more robust manner than the traditional Simple Certificate Enrollment Protocol (SCEP). There are open-source EST implementations for anyone to use and improve. This document describes the installation and implementation of open-source EST clients that you can experiment with or use to import EST into your products.

Contents

Introduction	2
Installation and Use	2
libEST	3
Python EST Client	4
Conclusion	6
References	6
Acknowledgment	7

Introduction

EST is an [IETF standard](#) that “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 went through several iterations through the [IETF](#). Multiple vendors and independent parties in the standards community participated in the effort.

EST is the successor to SCEP, initially sponsored by Cisco. SCEP has become the standard in certificate provisioning, but even though it is widely used, it is not standardized. (Recently it was [taken up again by IETF](#) for standardization, but a lack of area director support makes standardization unlikely.) On the other hand, on top of the high community participation in its development, EST also offers a variety of security advantages over SCEP that include elliptic curve cryptography (ECC) certificates and open-source code availability. Cisco itself has open-sourced [libEST](#), a library that offers EST client and server functionality, in order to promote its adoption and interoperability across vendors.

EST has several advantages:

- It uses TLS for the highly secure transport of certificates and messages.
- The transport is tied to the request. The certificate signing request (CSR) can be tied to a requestor that is already trusted and authenticated, so the certificate requestor owns the private key and the certificate is provided only to the entity requesting it.
- There are open-source implementations of EST for vendors and private parties to experiment with and adopt.
- It supports ECC certificates.
- It supports certificate reenrollment.

The issue of certificate provisioning and PKI is ubiquitous. Other communities such as OpenStack are addressing it with similar and different approaches. We believe that EST is the best candidate solution because of the openness of its development, the open-source code available, and the advantages it offers. For more information on EST and its advantages, refer to the [PKI: Simplify Certificate Provisioning with EST whitepaper](#).

The sections below present the basic steps of using the existing EST client software. Developers can follow these to install and use the existing open-source libraries. A tool we do not present in this document is [jester](#), a Java implementation of EST. `jester` includes the EST client and server, but falls beyond the scope of this document. As stated in its page, “`jester` aims to be 100% compatible with Cisco’s [libest](#) implementation.” Cisco did not write or test `jester` and doesn’t own or support it.

Installation and Use

The instructions below were performed using a 64-bit CentOS 7 virtual machine. They should work the same in all Red Hat–based systems and very similarly in others like Ubuntu. The goal of the steps is to take a new user through the process of getting the open-source EST libraries working against an EST server. We do not demonstrate the full EST protocol, but by following the steps you should be able to make full use of the options that the libraries offer in order to use an EST client with an EST server. For our case, we use the Cisco® public EST server located at <http://testrfc7030.cisco.com>. For more information on how to use the server in `libEST`, refer to the EST server section of the Cisco [IOS EST client example whitepaper](#).

libEST

Cisco maintains the open-source library [libEST](#). It is a portable library that implements RFC 7030 (EST). It is written in C and uses OpenSSL 1.0 and can be used to provision certificates from a certificate authority (CA) or registration authority (RA).

1. First download libEST from [GitHub](#):

```
git clone https://github.com/cisco/libest.git
```

2. Then untar the tar.gz file, and compile and install libEST:

```
tar xvf libest.tar.gz && cd libest
make clean && ./configure --with-ssl-dir=/usr/local/openssl1.0.1p && make
sudo make install
```

3. Run the example test client:

```
export LD_LIBRARY_PATH=/usr/local/openssl1.0.1p/lib:/usr/local/est/lib
wget http://testrfc7030.cisco.com/qvrca2.pem
export EST_OPENSSL_CACERT=./qvrca2.pem
./example/client-simple/estclient_simple -s testrfc7030.cisco.com -p 8443 \
-u estuser -h estpwd
```

The `example/client-simple/estclient_simple` example program showcases four EST requests. First the `/cacerts` request is sent to the server. Next, the `/csrattrs` request is sent to the server. Then the application creates a certificate signing request (CSR) and then CSR is sent to the server using `/simpleenroll` and an X509 certificate is returned. libEST supports the `/simplereenroll` function as well.

`estuser` and `estpwd` are the default credentials to authenticate the EST client against the EST server. The `qvrca2.pem` certificate downloaded from the server is the implicit CA certificate that the EST client is using to verify the certificate offered by the server in the HTTPS connection in order to authenticate it. Moreover, without setting the right path in `LD_LIBRARY_PATH` pointing to the OpenSSL library (`/usr/local/openssl1.0.1p/lib` for my system) libEST will not compile and give an error like

```
/home/panos/libest-1.2.0/example/client-simple/.libs/lt-estclient_simple: error
while loading shared libraries: libssl.so.1.0.0: cannot open shared object file: No
such file or directory
```

After the example program is run, three files are created in the current directory:

- `newcacerts.pkcs7` contains the latest copy of the CA certificates from the EST server.
- `cert-b64.pkcs7` contains the newly provisioned certificate.
- `new_key.pem` contains the new key pair that was generated when creating the certificate request. It contains the private key that matches the public key in the new certificate.

To view the new CA certificates returned by the server and the newly provisioned certificate, you could use

```
openssl base64 -d -in ./newcacerts.pkcs7 | \
```

```
openssl pkcs7 -inform DER -text -print_certs
openssl base64 -d -in ./cert-b64.pkcs7 | \
openssl pkcs7 -inform DER -text -print_certs
```

The `libest/example` directory has more example programs. For example, `libest/example/client` shows you how to use an existing key in the CSR request, how to retrieve the CSR attributes from the server separately, and how to authenticate the client using an already provisioned certificate.

Although it falls outside the scope of this white paper, readers should note that the libEST library can also be used for EST server functionality. `est/example/server` has a program, and the README explains how to set it up. For more information on how to use the server in libEST, refer to the EST server section of the Cisco [IOS EST client example whitepaper](#).

Note about OpenSSL: If you are using OpenSSL 1.0.1e or earlier, libEST might not compile because of missing header files like (`include/srp.h`). Thus someone would need to upgrade OpenSSL to a later (and more secure) version.

After downloading the new OpenSSL version (let's say in `openssl1.0.1p.tar.gz`), you can untar and install it:

```
tar xvf openssl1.0.1p.tar.gz
cd openssl1.0.1p
make clean && ./config --openssldir=/usr/local/openssl1.0.1p shared
make && make test
sudo make install
```

If OpenSSL is not compiled with the `shared` flag, compilation errors might occur like

```
/usr/local/ssl/lib/libssl.a(s23_srvr.o): relocation R_X86_64_32 against `'.rodata'
can not be used when making a shared object; recompile with -fPIC
/usr/local/ssl/lib/libssl.a: could not read symbols: Bad value
```

Python EST Client

[est-client-python](#) is a Python-based tool that can act as an EST client. The EST operations it supports (as of September 2015) are `/simpleenroll`, `/simplereenroll`, and `/cacerts`. It requires Python packages `pyopenssl` and `requests`. The instructions below cover installation and use of the package with Python 2.7.10.

1. Before proceeding with the installation, make sure to install all the required Python packages `epel-release`, `python-pip`, `python-devel`, `libffi-devel`, and `openssl-devel` using

```
sudo yum install <package_name>
```

The last three packages are because of dependencies of `pyopenssl` and `requests`.

2. Then download the Python EST client:

```
git clone https://github.com/laurentluce/est-client-python.git
```

3. And download the EST server implicit CA certificate:

```
wget http://testrfc7030.cisco.com/qvrca2.pem -O server.pem
```

4. Run the test Python program:

```
cp est/test/live.py test-est-client.py
python2.7.10 test-est-client.py
```

The `test-est-client.py` example program showcases three EST requests. It first creates a CSR. Then it sends a `/cacerts` request to the server. Then the application sends the CSR to the server using `/simpleenroll` and an X509 cert is returned. Finally it sends a `/simplereenroll`. The client uses `estuser` and `estpwd` as the default credentials to authenticate the EST client against the EST server. The `server.pem` certificate downloaded from the server is the implicit CA certificate that the EST client is using to verify the certificate offered by the server in the HTTPS connection in order to authenticate it. In `test-est-client.py`, the file in the `“implicit_trust_anchor_cert_path = ”` line needs to contain the implicit certificate file `server.pem`. Note that if someone wanted to print the certificates generated, he could do so by printing out the `pem` variable in `est/client.py` file.

Alternatively, someone could create a file that uses the client Python function and run it. He could put the following code in a python file and run it as in Step 4:

```
import est.client

host = 'testrfc7030.cisco.com'
port = 8443
"""server.pem is the server implicit CA cert downloaded with
   wget http://testrfc7030.cisco.com/qvrca2.pem -O server.pem
   """
implicit_trust_anchor_cert_path = 'server.pem'

client = est.client.Client(host, port, implicit_trust_anchor_cert_path)
ca_certs = client.cacerts()

username = 'estuser'
password = 'estpwd'
client.set_basic_auth(username, password)

common_name = 'test'
country = 'US'
state = 'Massachusetts'
city = 'Boston'
organization = 'Cisco Systems'
organizational_unit = 'ENG'
email_address = 'test@cisco.com'
private_key, csr = client.create_csr(common_name, country, state, city,
                                     organization, organizational_unit,
                                     email_address)

client_cert = client.simpleenroll(csr)
client_cert = client.simplereenroll(csr)
```

Note about support: Even though it is very useful, Cisco did not officially write and doesn't own or support [est-client-python](#).

Note about Python: Versions of Python earlier than 2.7.10 might throw errors. Initially, version 2.7.5 was giving `InsecurePlatformWarning` warnings about an untrusted server certificate. The [reference](#) of the warnings suggested upgrading Python.

Someone has to install python 2.7.10 in a nondefault location (`/usr/local`) to not break the existing Python installation. Let's download the `python2.7.10.tar.gz` file and install it using

```
tar xvf python2.7.10.tar.gz && cd Python2.7.10
make clean && ./configure --prefix /usr/local/ && make
./python -V (to make sure it compiled)
sudo make install
ln -s /usr/local/bin/python /usr/bin/python2.7.10
python2.7.10 -V (to make sure the symbolic link works)
```

Then try to run the Python test. If it doesn't work because of a missing OpenSSL package imported by `est/client.py`, find the package locations using

```
pip list
pip show pyopenssl
pip show requests
```

The packages are in the "system pre-installed" (version 2.7.5) Python's `/usr` location. Thus, you should install the packages in the new Python 2.7.10 installation. First download `get-pip.py` and install pip:

```
sudo python2.7.10 get-pip.py
```

pip (for Python 2.7.10) gets installed in `/usr/local/bin/pip` and not in `/usr/bin/pip` of the Python system installation (2.7.5). Then install the two packages in the new Python 2.7.10 version:

```
sudo /usr/local/bin/pip install pyopenssl
sudo /usr/local/bin/pip install requests
```

Finally, the test program in Step 4 can be run successfully.

Conclusion

The EST libraries presented here are very portable and easy to use. As time goes by, more clients and public certificate authorities will adopt EST. The open-source code will make it easier to quickly bring EST into more and more products, so vendors can use this common protocol in a modern and efficient manner.

References

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

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

est-client-python: <https://github.com/laurentluce/est-client-python>

SCEP: <https://tools.ietf.org/id/draft-gutmann-scep>

Acknowledgment

Panos Kampanakis (panosk[at]cisco[dot]com)

Technical Marketing Engineer