Provisioning X.509 Certificates Using RFC 7030

Learn how to use libest to deploy X.509 certificates across your enterprise.

JOHN FOLEY

Have you ever found yourself in the need of an X.509 certificate when configuring a Linux service? Perhaps you're enabling HTTPS using Apache or NGINX. Maybe you're configuring IPsec between two Linux hosts. It's not uncommon to use on-line forums or developer blogs to find setup instructions to meet these goals quickly. Often these sources of information direct the reader to use a self-signed certificate. Such shortcuts overlook good security practices that should be followed as part of a sound Public Key Infrastructure (PKI) deployment strategy. This article proposes a solution to the problem of widespread deployment of X.509 certificates using Enrollment over Secure Transport (EST). First, I provide a brief primer on PKI. Then, I give an overview of using EST to provision a certificate, demonstrated using both curl and libest. Finally, I show a brief example of an OpenVPN deployment using certificates provisioned with EST.

PKI Primer

Public Key Infrastructure consists of several building blocks, including X.509 certificates, Certificate Authorities, Registration Authorities, public/private key pairs and certificate revocation lists. X.509 certificates are provisioned by end-entities from either an RA or a CA. An end-entity will use the X.509 certificate to identity itself to a peer when establishing a secure communication channel. The X.509 certificate contains a public key that another entity can use to verify the identity of the entity presenting the X.509 certificate. The owner of an X.509 certificate retains the private key associated with the public key in its X.509 certificate. The private key is used only by the end-entity that owns the X.509 and must remain confidential. Leakage of the private key compromises the security model.

X.509 certificates are the most commonly used approach for verifying peer identity on the Internet today. X.509 certificates are used with TLS, IPsec, SSH and other protocols. For example, a Web browser will use the X.509 certificate from a Web server to verify the identity of the server. This is achieved by using the public key in the CA certificate and the signature in the Web server certificate.

PKI allows for multiple layers of trust, with the root CA at the top of the trust chain. The root CA also is called a trust anchor. The root CA can delegate authority to a sub-CA or an RA. The sub-CA or RA can provision X.509 certificates on behalf of an end-entity, such as a

Web browser or a Web server. For simplicity, this article is limited to showing a single layer of trust where the root CA generates certificates directly for the end-entities.

The multiple layers of trust in the PKI hierarchy are implemented using asymmetric cryptographic algorithms. RSA is the most common asymmetric algorithm used today. RSA is named after its inventors: Ron Rivest, Adi Shamir and Leonard Adleman. RSA uses a public/private key pair. The X.509 certificate generated by a CA contains a digital signature. This signature is the encrypted output from the RSA algorithm. The CA will calculate the hash (for example, SHA1) of the certificate contents. The hash value is then encrypted using the CA's private RSA key. The CA also will include information about itself in the new certificate in the Issuer field, which allows another entity to know which CA generated the certificate (Figure 1). This becomes

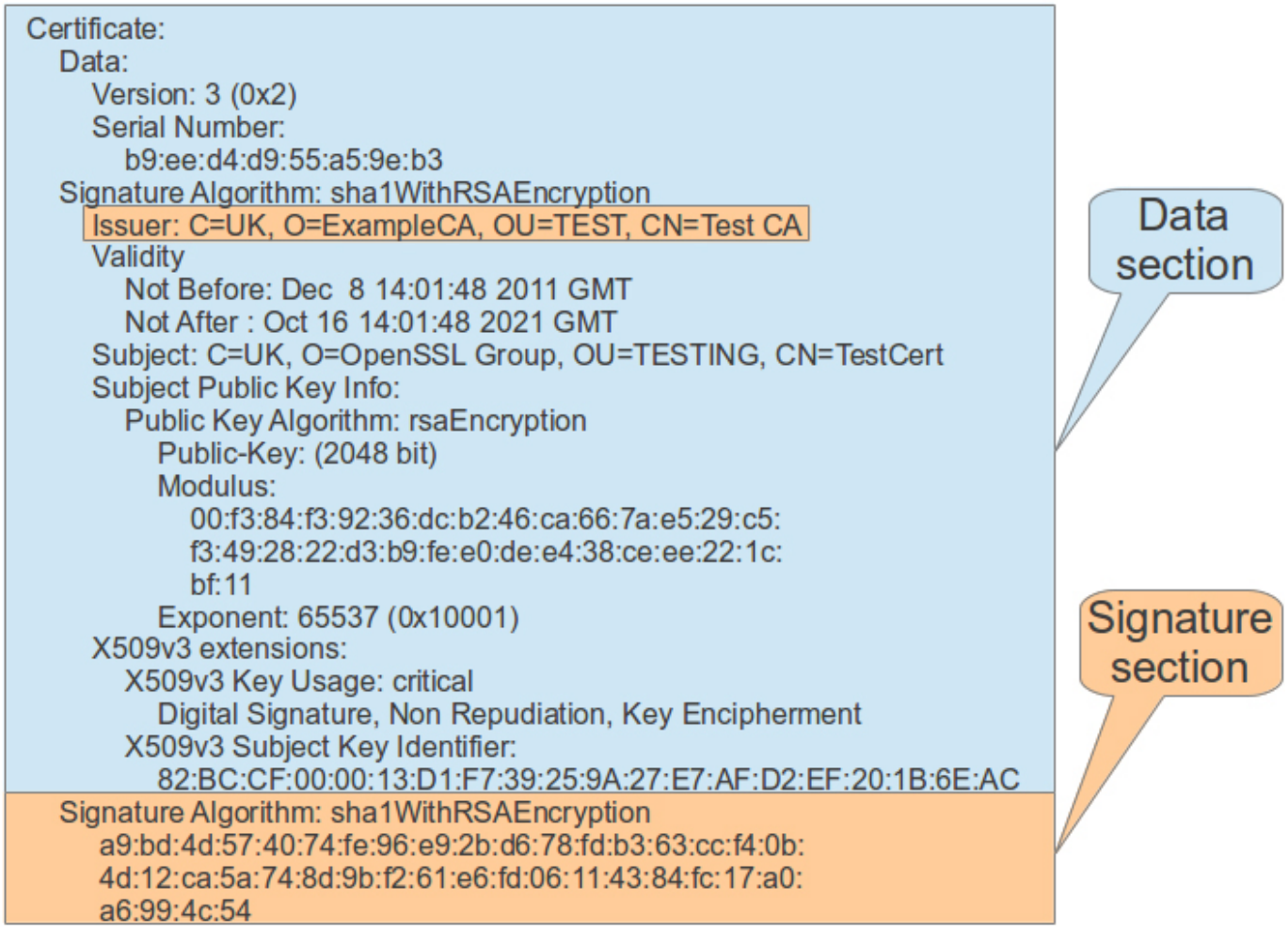

Figure 1. Anatomy of an X.509 Certificate

important when another entity needs to verify the authenticity of another entity's certificate.

When another entity needs to verify the identity and authenticity of an X.509 certificate, the public key from the CA certificate is used to verify the X.509 certificate to be verified. The verifying identity calculates the hash (for example, SHA1) of the certificate, similar to how the CA did this when generating the certificate. Instead of using the CA private key, the verifying entity will use the CA public key to encrypt the hash result. If the public key encrypted value of the hash matches the signature that is in the X.509 certificate, the verifying identity is assured that the certificate is valid and was issued by the CA.

Astute readers will observe that the verifying entity needs to have the CA public key to perform this verification process. This problem is commonly solved by deploying the public keys of well-known certificate authorities with the software that will be performing the verification process. This is known as out-of-band trust anchor deployment. Commonly used Web browsers (such as Firefox, Chrome and IE) follow this model. When you install the Web browser on your computer, the well-known CA certificates are installed with the software. When you browse to a secure Web site, the Web browser uses this trust anchor of pre-installed CA certificates

Elliptic Curve Digital Signature Algorithm

Elliptic Curve Digital Signature Algorithm (ECDSA) is an asymmetric cryptographic algorithm that can be used similar to RSA. ECDSA uses a public/private key pair similar to RSA. However, ECDSA provides an equivalent security level to RSA using much smaller key sizes. As of 2014, the NIST minimum recommended RSA key size is 2048 bits. Using an ECDSA 256-bit key provides better security than RSA 2048, which significantly reduces the amount of data that needs to be transmitted between peers during the key exchange process of protocols like TLS, DTLS and IKE. This bandwidth savings is desirable for IoT devices or other end-entities that need to minimize bandwidth consumption over radio interfaces.

FQDN

X.509 certificates should contain the Fully Qualified DNS Domain Name (FQDN) of the entity that owns the certificate. RFC 6125 provides guidance on how FQDN should be implemented in a PKI deployment. The FQDN should be in either the Subject Common Name or the Subject Alternative Name of the X.509 certificate. When an entity, such as a Web browser, verifies the authenticity of the peer certificate, the FQDN should be checked to ensure that it matches the hostname used to initiate the IP connection. For example, when you browse to https://www.foobar.org, the X.509 certificate presented by the Web server should contain www.foobar.org in either the Subject Common Name or Subject Alternative Name. Checking the FQDN helps mitigate an MITM (Man In The Middle) attack.

(containing public keys) to verify the X.509 certificate presented by the Web server.

Enrollment over Secure Transport

Enrollment over Secure Transport (EST) is defined in RFC 7030. This protocol solves the challenge of PKI deployment across a large infrastructure. For example, RFC 7030 defines methods for both provisioning end-entity certificates and deploying CA public keys, which are required for end-entities to verify each other. This article focuses on the client side of solving these two challenges. Let's use the interop test server hosted at **http://testrfc7030.cisco.com** as the EST server for the examples. *Note: the certificates generated by this test server are for demonstration purposes only and should not be used for production deployments.*

RFC 7030 defines a REST interface for various PKI operations. The first operation is the /cacerts method, which is used by an end-entity to retrieve the current CA certificates. This set of CA certificates is called the *explicit* trust anchor. The /cacerts method is the first step invoked by the end-entity to ensure that the latest trust anchor is used for subsequent EST operations. The following steps show how to use curl as an EST client to issue the /cacerts operation.

Step 1: Retrieve the public CA certificate used by testrfc7030.cisco.com. Note: this step emulates the out-of-band deployment of the *implicit*

trust anchor:

```
wget http://testrfc7030.cisco.com/DST_Root_CA_X3.pem
```

Step 2: Use curl to retrieve the latest *explicit* trust anchor from the test server:

```
curl https://testrfc7030.cisco.com:8443/.well-known/est/cacerts \
    -o cacerts.p7 --cacert ./DST_Root_CA_X3.pem
```

Note: you can use a Web browser instead of wget/curl for steps 1 and 2. Enter the URL shown in step 2 into your browser. Then save the result from the Web server to your local filesystem using the filename cacerts.p7.

Step 3: Use the OpenSSL command line to convert the trust anchor to PEM format. This is necessary, because the EST specification requires the /cacerts response to be base64-encoded PKCS7. However, the PEM format is more commonly used:

```
openssl base64 -d -in cacerts.p7 | \
   openssl pkcs7 -inform DER -outform PEM \
   -print_certs -out cacerts.pem
```

The certificate in cacerts.pem is the explicit trust anchor. You will use this certificate later to establish a secure communication channel between two entities. However, first you need to provision a certificate for each entity. You'll use OpenSSL to create a certificate request. The certificate request is called a CSR, defined by the PKCS #10 specification. You'll also create your public/private RSA key pair when creating the CSR. You'll use OpenSSL to create both the CSR and the key pair.

Step 4: Generate an RSA public/private key pair for the end-entity and create the CSR. The CSR will become the X.509 certificate after it has been signed by the CA. You will be prompted to supply the values for the Subject Name field to be placed in the X.509 certificate. These values include the country name, state/province, locality, organization name, common name and your e-mail address. The common name should contain the FQDN of the entity that will use the certificate. The challenge password and company name are not required and can be left blank:

```
openssl req -new -sha256 -newkey rsa:2048 \
    -keyout privatekey.pem -keyform PEM -out csr.p10

Generating a 2048 bit RSA private key
........+++
.....................................................
.....................................................
.................+++
writing new private key to 'privatekey.pem'
```

```
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that
will be incorporated
into your certificate request.
What you are about to enter is what is called a
Distinguished Name or a DN.
There are quite a few fields but you can leave some
blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Colorado
Locality Name (eg, city) []:Aspen
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ACME, Inc.
Organizational Unit Name (eg, section) []:Mfg
Common Name (e.g. server FQDN or YOUR name) []:mfgserver1.acme.org
Email Address []:jdoe@acme.org

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Now that you have the CSR and key pair, you need to send the CSR to the CA to have it signed and returned to you as an X.509 certificate. The EST /simpleenroll REST method is used for this purpose. Curl can be used again to send the CSR to the CA as a RESTful EST operation.

Step 5: Use curl to enroll a new certificate from the test server using the CSR you just generated. Normally the explicit trust anchor is used for this step. However, the test server doesn't use the explicit trust anchor for HTTPS services. Therefore, you'll continue to use the implicit trust anchor (DST_Root_CA_X3.pem) for this step (note: the test CA at testrfc7030.cisco.com uses a well-known user name/password of estuser/estpwd):

```
curl https://testrfc7030.cisco.com:8443/.well-known/est/simpleenroll \
    --anyauth -u estuser:estpwd -o cert.p7 \
    --cacert ./DST_Root_CA_X3.pem --data-binary @csr.p10 \
    -H "Content-Type: application/pkcs10"
```

Step 6: If successful, the curl command should place the new certificate in the cert.p7 file. The EST specification requires the certificate to be base64-encoded PKCS7. Because PEM is a more commonly used format, you'll use OpenSSL to convert the new certificate to PEM format:

```
openssl base64 -d -in cert.p7 | openssl pkcs7 -inform DER \
    -outform PEM -print_certs -out cert.pem
```

Step 7: Finally, use OpenSSL again to confirm the content in the certificate. The Subject Name should contain the values you used to create

the CSR earlier:

```
openssl x509 -text -in cert.pem

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 42 (0x2a)
  Signature Algorithm: ecdsa-with-SHA1
    Issuer: CN=estExampleCA
    Validity
        Not Before: Jun  4 18:42:56 2014 GMT
        Not After : Jun  4 18:42:56 2015 GMT
    Subject: CN=mfgserver1.acme.org
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:c0:4c:65:d1:6c:d2:8b:7d:37:b9:a1:67:da:7a:
                a1:6c:4f:b9:9f:68:e0:9a:44:24:a0:aa:54:55:19:
                c0:fc:6b:35:c5:a7:14:ed:70:e9:99:32:6a:21:19:
                49:2b:8e:42:89:eb:9f:ec:3d:69:75:49:2f:f7:18:
                f6:14:ed:d5:71:54:b5:0a:d0:f3:7b:8e:36:19:f1:
                45:07:37:b9:aa:73:7c:60:bb:e1:f1:ac:b2:75:74:
                22:9e:5d:b5:ee:13:7c:b8:31:61:c5:9a:ef:7e:07:
                24:8d:c8:50:44:89:6d:fe:dd:e0:28:fd:80:1c:b9:
                61:94:8d:63:cd:54:2c:a9:86:7a:3b:35:62:e9:c6:
                76:58:fb:27:c1:bf:db:c2:03:66:e5:dd:cb:75:bc:
                72:6c:ca:27:76:2a:f7:48:d5:3b:42:de:85:8e:3b:
                15:f1:7a:e4:37:3c:96:b2:91:70:6f:97:22:15:c6:
                82:ea:74:8b:f2:80:39:c1:c2:10:78:6e:70:11:78:
                31:2f:4a:c3:c4:2b:ab:2f:4d:f2:87:15:59:88:b3:
                17:12:1d:92:b2:6d:a6:8a:94:3f:b3:76:18:53:f9:
                59:29:e1:9b:8c:81:41:7e:8c:a2:a7:34:c9:b4:07:
                32:77:57:37:59:dd:fb:36:02:59:74:bb:96:6e:e7:
                3f:b7
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 Key Usage:
            Digital Signature
        X509v3 Subject Key Identifier:
      E2:C5:FC:55:42:D9:52:D9:81:F7:CC:6C:01:56:BF:10:35:41:7A:D8
        X509v3 Authority Key Identifier:
      keyid:EE:DE:AA:C0:5B:AC:38:7D:F3:08:26:33:73:00:3F:F3:2B:63:41:F8

  Signature Algorithm: ecdsa-with-SHA1
      30:45:02:20:1e:b6:b6:32:fa:79:de:26:c0:34:0d:a5:5c:70:
      cb:27:a3:8f:fc:9f:d2:1f:ca:5c:99:fd:d0:ff:bf:7f:51:e8:
      02:21:00:be:1f:36:b3:f6:46:65:58:eb:57:05:c3:af:4c:4a:
      0e:d1:28:e9:0b:58:e3:ac:3f:db:27:36:33:98:3f:b1:9e
```

At this point, you have the new certificate and associated private key. The certificate is in the file cert.pem. The private key is in the file privatekey.pem. These can be used for a variety of security protocols including TLS, IPsec, SSH and so on.

libest

Curl provides a primitive method to issue the RESTful operations of the EST enrollment process. However, the curl command-line options required to enroll a new certificate securely are cumbersome and error-prone. Additionally, curl is unable to perform the TLS channel binding

requirements defined in RFC 7030 section 3.5. There is an open-source alternative called libest. This library supports client-side EST operations required to provision a certificate. The libest library comes with a client-side command-line tool to replace the curl commands described earlier. Additionally, libest exposes an API when EST operations need to be embedded into another application. Next, I demonstrate how use the libest CLI to enroll a certificate from the test server.

libest is available at **https://github.com/cisco/libest**. It's known to work on popular Linux distributions, such as Ubuntu and Centos. You will need to download, configure, compile and install libest to follow along. The default installation location is /usr/local/est. libest requires that you install the OpenSSL devel package prior to configuration. OpenSSL 1.0.1 or newer is required.

You'll use the same implicit trust anchor and CSR that you used earlier when using curl as the EST client. The implicit trust anchor is located in DST_Root_CA_X3.pem, and the CSR is in csr.p10.

Step 1: Configure the trust anchor to use with libest:

```
export EST_OPENSSL_CACERT=DST_Root_CA_X3.pem
```

Step 2: Using the same CSR used earlier with curl in the csr.p10 file, provision a new X.509 certificate for the CSR:

```
estclient -e -s testrfc7030.cisco.com -p 8443 \
    -o . -y csr.p10 -u estuser -h estpwd
```

Heartbleed

The Heartbleed bug in OpenSSL was a severe vulnerability in OpenSSL that was publicly announced in April 2014. The severity of this bug was due to the potential for leaking the X.509 private key of the TLS server. When a private key is leaked, previously recorded communications using the key can be compromised. For example, a compromised TLS server may have had all communications revealed since the private key was issued to it. It's not uncommon for certificates to be issued for a year or longer. Imagine every transaction you conducted with your on-line bank during the past year being compromised. The Heartbleed bug provides motivation to use shorter validity periods when issuing X.509 certificates. EST can be used to automate the certificate renewal process to avoid interruptions in service due to certificate expiry.

Step 3: Similar to the curl example shown earlier, use OpenSSL to convert the new certificate to PEM format and confirm the contents of the certificate:

```
openssl base64 -d -in ./cert-0-0.pkcs7 | \
    openssl pkcs7 -inform DER  -print_certs -out cert.pem
openssl x509 -text -in cert.pem
```

This enrollment procedure can be used on any number of end-entities. These end-entities than can use their certificate along with the explicit trust anchor to verify each other when establishing secure communications. This eliminates the need to generate self-signed certificates and manually copy those certificates among end-entities. The enrollment process could be automated using curl or libest. EST can be used for certificate renewal as well, which can automate the process of renewing certificates that are about to expire. Automating the process can facilitate the shortening of certificate validity periods, which improves the overall security posture of a PKI deployment.

Using the New Certificates

Now that you have provisioned a new certificate for the end-entity, the certificate can be used with a variety of protocols. Next, I show how the certificate can be used with OpenVPN to establish a secure communication channel between two Linux hosts. The certificate enrollment process described earlier needs to be completed on each Linux host.

OpenVPN supports a wide variety of configurations. In this example, let's use the TLS client/server model. Two Linux hosts are required. OpenVPN should be installed on both systems. One host will operate as the TLS server for VPN services. The other host will operate as the TLS client. In this example, the IP address for the physical Ethernet interface on the server is 192.168.1.35. The TAP

Listing 1. TLS Server OpenVPN Config

```
dev tap
ifconfig 10.3.0.1 255.255.255.0
tls-server
dh dh2048.pem
ca cacerts.pem
cert cert.pem
key privatekey.pem
```

Listing 2. TLS Client OpenVPN Config

```
remote 192.168.1.35
dev tap
ifconfig 10.3.0.2 255.255.255.0
tls-client
ca cacerts.pem
cert cert.pem
key privatekey.pem
```

interface is used on both the client and the server for the VPN tunnel. The server TAP interface uses the address 10.3.0.1, while the client uses 10.3.0.2. The certificates provisioned using EST are configured in the OpenVPN configuration file on each system (see Listings 1 and 2).

Step 1: Generate DH parameters for the OpenVPN server:

```
openssl gendh -out dh2048.pem 1024
```

Step 2: Start the OpenVPN server:

```
sudo openvpn vpnserver.conf
```

Step 1: Start the OpenVPN client:

```
sudo openvpn vpnclient.conf
```

Step 2: Ping across the tunnel to ensure that the VPN is working:

```
ping 10.3.0.1
```

Summary

This article has focused on the minimal client-side EST operations required to establish an explicit trust anchor and provision a new certificate. EST provides additional capabilities including certificate renewal, CSR attributes and server-side key generation. EST also provides for various client authentication methods other than using a user name/password. The client can be authenticated using SRP or an existing X.509 certificate. For example, an existing certificate on the EST client should be used when renewing a certificate prior to expiration. A good source of information for learning more about these concepts is the EST specification (RFC 7030).

This article has focused on the client side. In the future, I will look at implementing EST on the server side to front a CA. The EST protocol is a new protocol and not widely adopted at this time. Several well-known commercial CA vendors are currently implementing EST. If you use a commercial CA as part of your PKI infrastructure today, you may want to ask your CA vendor about its plans to support EST in the future.■

John Foley is a Technical Leader in product development at Cisco Systems. John has worked on a variety of projects during the past 14 years at Cisco, including VoIP systems, embedded development, security and multicast. John has spent the past three years working with security protocols, including TLS, SRTP and EST. Prior to this, John worked for seven years in the IT industry doing application development and RDBMS design. John is an active contributor to the libest and libsrtp projects on GitHub.

Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.