Setting up a Certificate Authority

During development, it is convenient to have a simple way of creating new certificates. OpenSSL includes all of the necessary infrastructure for setting up your own certificate authority (CA), but it requires getting more familiar with OpenSSL than is really necessary. To simplify the process, Ice includes the Python script `iceca`, located in the `bin` subdirectory of your Ice installation, that hides the complexity of OpenSSL and allows you to quickly perform the essential tasks:

- initializing a new root CA
- generating new certificate requests
- signing certificate requests to create a valid certificate chain
- converting certificates to match platform-specific requirements.

You are not obligated to use this script; IceSSL accepts certificates from any source as long as they are provided in the appropriate formats. However, you may find this tool sufficient for your development needs, and possibly even for your deployed application as well.

On this page:

Initializing a Certificate Authority

Some of the script's activities use a directory that contains configuration files and a database of issued certificates. The script selects a default location for this directory that depends on your platform, or you can specify the parent directory explicitly by defining the `ICE_CA_HOME` environment variable and the script will use `$ICE_CA_HOME/ca` for its files.

The script command `iceca init` initializes a new CA by preparing a database directory and generating the root CA certificate and private key. It accepts the following command-line arguments:

```
$ python iceca init [--no-password] [--overwrite]
```

Upon execution, the script first checks the database directory to determine whether it has already been initialized. If so, the script terminates immediately with a warning unless you specify the `--overwrite` option, in which case the script overwrites the previous contents of the directory.

Next, the script displays the database directory it is using and begins to prompt you for the information it needs to generate the root CA certificate and private key. It offers a default choice for the CA's distinguished name and allows you to change it:

```
The subject name for your CA will be
CN=Grid-CA ,  O=GridCA-server
Do you want to keep this as the CA subject name? (y/n) [y]
```

To specify an alternate value for the distinguished name, enter n and type the new information, otherwise hit Enter to proceed.

```
Enter the email address of the CA: ca-admin@company.com
```

The address you provide in response to this prompt is shown to users that create certificate requests. Enter the address to which such requests should be sent.

The script shows its progress as it generates the certificate and private key, then prompts you for a pass phrase. If you prefer not to secure your CA's private key with a pass phrase, use the `--no-password` option when starting the script.

Upon completion, the script emits the following instructions:

```
The CA is initialized.

You need to distribute the following files to all machines that
can request certificates:

C:\iceca\req.cnf
C:\iceca\ca_cert.pem

These files should be placed in the user's home directory in
~/.iceca. On Windows, place these files in <ice-install>/config.
```

In this example, the `ICE_CA_HOME` environment variable was set to `C:\iceca`. As the script states, the files `req.cnf` and `ca_cert.pem` must be present on each host that can generate a certificate request. The script suggests a location for these files, which is the default directory used by the scripts if `ICE_CA_HOME` is not defined.

The `ca_cert.pem` file contains the root CA's certificate. Your IceSSL configurations must identify this certificate (in its proper form for each platform) as a trusted certificate. For example, you can use this file directly in the configuration of the C++ plug-in:

```
IceSSL.CertAuthFile=C:\iceca\ca_cert.pem
```

For .NET applications, you should import the certificate file into the proper store.

In Java, you need to add the certificate to your truststore:

```
$ keytool -import -trustcacerts -file ca_cert.pem -keystore ca.jks
Enter keystore password:
```

The keytool program requires you to enter a password, which you could use as the value of the property `IceSSL.TruststorePassword`.

Now that your certificate authority is initialized, you can begin generating certificate requests.

Generating Certificate Requests

The script command `iceca request` uses the files you created while initializing the CA to generate a request for a new certificate. It accepts the following command-line arguments:

```
$ python iceca request [--overwrite] [--no-password] file common-name [email]
```

The script looks for the files `req.cnf` and `ca_cert.pem` in the directory defined by the `ICE_CA_HOME` environment variable. If that variable is not defined, the script uses a default directory that depends on your platform.

The purpose of the script is to generate two files: a private key and a file containing the certificate request. The request file must be transmitted to the certificate authority for signing, which produces a valid certificate chain.

The argument `file` is used as a prefix for the names of the two output files created by the script:

- *`file`*`_key.pem` contains the private key
- *`file`*`_req.pem` contains the certificate request

If the output files already exist, you must specify `--overwrite` to force the script to overwrite them.

The `common-name` argument defines the common name component of the certificate's distinguished name. If the optional `email` argument is provided, it is also included in the certificate request.

During execution, the script displays its progress as it generates the necessary files. It will prompt you for a pass phrase unless you used the `--no-password` option, and finish by showing the names of the files it created as well as instructions on how to proceed. The example below shows the output from generating a request for an IceGrid node using a filename prefix of `node`:

```
$ iceca request node "IceGrid Node"

Created key: node_key.pem
Created certificate request: node_req.pem

The certificate request must be signed by the CA. Send the
certificate request file to the CA at the following email
address:
ca-admin@company.com
```

The file `node_key.pem` is the new private key for the node; this file must be kept secure. The file `node_req.pem` must be given to the certificate authority. As a convenience, the script displays the CA's email address that you entered during initialization.

Signing Certificate Requests

As a certificate authority, you are responsible for certifying the validity of certificate requests by signing them with your private key. The product of signing a request is a valid certificate chain that a person or application can use as an identity. The `iceca sign` command performs this task for you and accepts the following command-line arguments:

```
$ python iceca sign [--overwrite] --in <req> --out <cert> [--ip <ip> --dns <dns>]
```

The input file `req` is the certificate request, and the output file `cert` is the resulting certificate chain. The script does not overwrite the file `cert` unless you also specify `--overwrite`. The `--{ip` and `--dns` options allow you to add subject alternative names to the certificate for IP and DNS addresses, respectively.

Continuing our previous example, we can sign the node's request with the following command:

```
$ python iceca sign --in node_req.pem --out node_cert.pem
```

If the CA's private key is protected by a pass phrase, we must enter that first. Next, the script displays the relevant information from the certificate request and asks you to confirm that you wish to sign the certificate:

```
The Subject's Distinguished Name is as follows
organizationName      :PRINTABLE:'Company.com'
commonName            :PRINTABLE:'IceGrid Node TestNode'
Certificate is to be certified until Jun 15 18:32:36 2011 GMT
Sign the certificate? [y/n]:
```

After reviewing the request, enter y to sign the certificate, and y again to finish the process. Upon completion, the script stores the certificate chain in the file `node_cert.pem` in your current working directory. This file, together with the node's private key we created when generating the request, establishes a secure identity for the node.

Importing Certificates

For Java and .NET users, the private key and certificate chain must be converted into a suitable format for your platform. The script command `iceca import` simplifies this process and accepts the following command-line arguments:

```
$ python iceca import [--overwrite] [--key-pass password] [--store-pass password]
  [--java alias cert key keystore] [--cs cert key out-file]
```

The script does not overwrite an existing file unless you specify `--overwrite`. To avoid interactive prompts for passwords, you can use the `--key-pass` option to specify the password for the private key, and the `--store-pass` option to define the password for the Java keystore. Completing our node example from prior sections, the command below imports the private key and certificate chain into a Java keystore:

```
$ python iceca import --java mycert node_cert.pem node_key.pem cert.jks
```

The value `mycert` represents the alias associated with this entry in the keystore, and `cert.jks` is the name of the new keystore file. In an IceSSL configuration, the property `IceSSL.Keystore` refers to this file.

The equivalent command for .NET is shown below:

```
$ python iceca import --cs node_cert.pem node_key.pem cert.pfx
```

The file `cert.pfx` uses the PKCS#12 encoding and contains the certificate chain and private key. You can import this certificate into a store, or refer directly to the file using the configuration property `IceSSL.CertFile`.

Certificate Authority Diagnostics

If you encounter a problem while using the `iceca` script, or simply want to learn more about the underlying OpenSSL commands used by the script, you can run the script with the `--verbose` option as shown below:

```
$ python iceca --verbose command ...
```

This option causes the script to display the commands as it executes them.

The script creates temporary files and directories that are normally deleted prior to the script's completion. If you would like to examine the contents of these files and directories, use the `--keep` option:

```
$ python iceca --keep command ...
```

See Also

- Configuring IceSSL