# RIoT Test Utilities

This solution contains two projects that

## RIoT.exe

A C# application that:

1. Creates test client and server RIoT certificate chains
2. Test-“vendor certifies” RIoT certificates generated by real RIoT implementations or the simulator
3. Implements a TLS test-server that can validate RIoT certificates
4. Provides command line parameter help for testing with OpenSSL s\_client and s\_server

## TlsClient.exe

A native code TLS/OpenSSL client test application that uses a RIoT certificate chain to authenticate to the server. It takes one or two parameters:

TlsClient [C|B] Directory

C Device should present a full certificate chain

B Device should present a “bare” Alias Certificate

By default, TlsClient looks for PEM-encoded keys and certificates in ./Certs (see below.)

# RIoT Usage

RIoT.exe supports several test configurations, summarized in the table below. Two processes are required to test the TLS handshake. The client and client instructions are offset in the order that they should be executed (i.e. start the server first.)

There are several supported scenarios. To do an end-end protocol tests, either:

RIoT -e2e

Or RIoT -bare -e2e

This creates Creates a new certificate set (-gentest), starts the test server (-server), and launches the native code TLS client (-tls\_test). If “-bare” then the device only presents the Alias Certificate; otherwise the device presents the whole chain up to the vendor CA.

Other options separate out certificate/CA generation from doing the protocol test. The two ways of creating a device certificate chain are:

**Create a dummy certificate set:**

RIoT -gentest

**“Vendor Certifying” a certificate bundle created by a RIoT device (see section below)**

RIoT -certify

Once a certificate set has been created, the following options can be used:

|  |  |  |
| --- | --- | --- |
| **Test Configuration** | **Server** | **Client** |
| RIoT server, TlsTest client | RIoT -server (or)  RIoT -bare -server | RIoT -tls\_client  (TlsClient.exe is started by RIoT.exe) |
| RIoT server, OpenSSL client | RIoT -server (or)  RIoT -bare -server | RIoT -ossl\_client |
| OpenSSL server, TlsTest client | RIoT -ossl\_server | RIoT -ossl\_client |

**Notes:**

1. The server and vendor PKI are randomly created on -gentest and -certify (and any previous files will be overwritten)
2. This all works best of RIoT.exe and TlsTest.exe are in the same directory. By default, the key and certificate files saved/loaded from ./Certs/…
3. The options -ossl\_client and -ossl\_server just print out a parameter set to start openssl with the correct configuration
4. RIoT -help for more options

# Certifying an Existing Device

RIoT -certify expects the following PEM-encoded files:

1. AliasCert.PEM
2. AliasKey.PEM
3. DeviceIDPublic.PEM

# Device Certificate Chain

The RIoT utility (and the client and server TLS test code) can create device certificate chains of arbitrary length. An example chain is illustrated below: if the chain is longer then there will be more Vendor Intermediate certificates. The device identity and state are expressed in the DeviceID certificate and the Alias Certificate, as described here:

## DeviceID Certificate

RIoT devices have a long-lived stable identity. The identity is encoded in a key pair called the DeviceID (we also use the term DeviceID when just referring to the public part of the identity.

The DeviceID certificate is signed using a key that is part of a vendor PKI. In the example this is a Vendor Intermediate Certificate.

The DeviceID key is used to sign the Alias Certificate.

## Alias Certificate

Devices use the Alias Key Pair and accompanying certificate to authenticate themselves. Important information is encoded a subject alt-name extension. This is a DER-encoded object containing:

1. The DeviceID
2. The Firmware ID (e.g. the firmware version number or hash.)

Note that the relying-party chain builder must verify the DeviceID in the Alias Certificate must match the DeviceID in the DeviceID certificate.)

Note also that a short version of the DeviceID is encoded in the DeviceID certificate and Alias Certificate. This is not designed to be an authoritative cryptographic identity: the alt-name extension authoritatively encodes this information.

# Files Produced and Consumed

By default, files are created and expected in the ./Certs directory from the current directory. Files are produced for use by both client and server, and there is some repetition of contents because different utilities need different sets of certificates grouped together.

**Device Certificates and Keys**

|  |  |
| --- | --- |
| **Name** | **Contents** |
| AliasCert.PEM | Just the alias certificate |
| AliasKey.PEM | Alias public and private key (not encrypted) |
| DeviceIDPublic.PEM | The public part of the DeviceID key pair |
| DeviceCertChain.PEM | The device certificate chain omitting the Alias Certificate. Order is important: first is the DeviceID certificate. Last is the Vendor CA certificate |
| DeviceCertChainIncAlias.PEM | As above, but including the Alias Certificate first in the list |
| DeviceCA.PEM | Just the device/vendor self-signed vendor-CA certificate |
|  |  |

**Server Certificate and Keys**

For the test utility, the server PKI is two certificates: a self-signed server-CA certificate, and a “server key” and certificate

|  |  |
| --- | --- |
| **Name** | **Contents** |
| ServerCert.PEM | Just the leaf server-certificate |
| ServerCA.PEM | Just the root cert |
| ServerChain.PEM | The root cert and leaf cert |
| ServerKey.PEM | The server key-pair |
| DeviceCA.PEM | Just the device/vendor self-signed vendor-CA certificate |

**Composite Files**

|  |  |
| --- | --- |
| **Name** | **Contents** |
| DeviceCertChainAndServerCA.PEM | Concatenation of DeviceCertChain.PEM and ServerCA.PEM |