Copyright (C) 2015-16 Jaguar Land Rover

This document is licensed under Creative Commons Attribution-ShareAlike 4.0 International.

CREATING RVI CERTIFICATES

This document describes how to generate the necessary certificates, keys and credentials needed for RVI Core. The example certificates are used in (rvi_protocol.md)[rvi_protocol.md].

STANDARDS USED

[1] JSON Web Token RFC7519- JWT (link)[https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32]
[2] base64url - (link)[https://en.wikipedia.org/wiki/Base64)
[3] Transport Layer Security (TLS) - (link)[https://en.wikipedia.org/wiki/Transport_Layer_Security]
[4] X.509 Certificates - (link)[https://en.wikipedia.org/wiki/X.509]

For all examples below the following certificates are used:

Sample root certificate

The self signed root certificate used in the examples throughout this document was generated using the following commands:

```
# Create root key pair
openssl genrsa -out insecure_root_key.pem 1024

# Create a self-signed root CA certificate, signed by the root key created above
openssl req -x509 -new -nodes -key insecure_root_key.pem -days 365 -out insecure
_root_cert.crt
```

The content of the sample `insecure_root_key.pem` private key file, which has no password protection, is:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDg5A1uZ5F36vQEYbMWCV4wY4OVmicYWEjjl/8YPA01tsz4x68i
/NnlMNalqpGCIZ0AwqGI5DZAWWoR400L3SAmYD6sWj2L9ViIAPk3ceDU8olYrf/N
wj78wVoG7qqNLgMoBNM584nlY4jy8zJ0Ka9WFBS2aDtB3Aulc1Q8ZfhuewIDAQAB
AoGAfD+C7CxsQkSc7I7N0q76SuGwIUc5skmUe6nOViVXZwXH2Or55+qqt+VzsbO7
EJphk7n0ZR0wm/zKjXd3acaRq5j3fOyXip9fDoNj+oUKAowDJ9vub0NOPpU2bgb0
xDnDeR0BRVBOTWqrkDeDPBSxw5RlJunesDkamAmj4VXHHgECQQDzqDtaEuEZ7x7d
kJKCmfGyP01s+YPlquDgogzAeMAsz17TFt8JS4RO0rX71+lmx7qqpRqIxVXIsR58
NI2Th7tRAkEA7Eh1C1WahLCxojQOam/l7GyE+2ignZYExqonOOvsk6TG0LcFm7W9
x39ouTlfChM26f8VYAsPxIrvsDlI1DDCCwJBAITmA8lzdrgQhwNOsbrugLg6ct63
kcuZUqLzgIUS168ZRJ1aYjjNqdLcd0pwT+wxkI03FKv5Bns6sGgKuhX3+KECQFm/
Z93HRSrTZpViynr5R88WpShNZHyW5/eB1+YSDslB1FagvhuX2570MRXxybys8bXN
sxPI/9M6prI8AALBBmMCQD+2amH2Y9ukJy10WuYei943mrCsp1oosWjcoMADRCpj
ZA2UwSzj67PBc5umDIAlhVRMX0zH/gLj54rfIkH5zLk=
-----END RSA PRIVATE KEY-----
```

The root key above is checked in as `priv/keys/insecure_root_key.pem`.

The content of the sample `insecure_root_cert.crt` file is:

```
-----BEGIN CERTIFICATE-----
MIICUjCCAbugAwIBAgIJAMI080XZPsPUMA0GCSqGSIb3DQEBCwUAMEIxCzAJBgNV
BAYTAlVTMQ8wDQYDVQQIDAZPcmVnb24xETAPBgNVBAcMCFBvcnRsYW5kMQ8wDQYD
VQQKDAZHRU5JVkkwHhcNMTUxMTI3MjMxMTQ0WhcNMTYxMTI2MjMxMTQ0WjBCMQsw
CQYDVQQGEwJVUzEPMA0GA1UECAwGT3JlZ29uMREwDwYDVQQHDAhQb3J0bGFuZDEP
MA0GA1UECgwGR0VOSVZJMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDg5A1u
Z5F36vQEYbMWCV4wY4OVmicYWEjjl/8YPA01tsz4x68i/NnlMNalqpGCIZ0AwqGI
5DZAWWoR400L3SAmYD6sWj2L9ViIAPk3ceDU8olYrf/Nwj78wVoG7qqNLgMoBNM5
84nlY4jy8zJ0Ka9WFBS2aDtB3Aulc1Q8ZfhuewIDAQABo1AwTjAdBgNVHQ4EFgQU
4Sz8rAMA+dHymJTlZSkap65qnfswHwYDVR0jBBgwFoAU4Sz8rAMA+dHymJTlZSka
p65qnfswDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQsFAAOBgQDFOapf3DNEcXgp
1u/g8YtBW24QsyB+RRavA9oKcFiIaHMkbJyUsOergwOXxBYhduuwVzQQo9P5nR0W
RdUfwtE0GuaiC8WUmjR//vKwakj9Bjuu73ldYj9ji9+eXsL/gtpGWTIlHeGugpFs
mVrUm0lY/n2ilJQ1hzBZ9lFLq0wfjw==
-----END CERTIFICATE-----
```

The root certificate above is checked in as `priv/certificates/insecure_root_cert.crt`.

DO NOT USE THE KEYS AND CERTIFICATES ABOVE IN PRODUCTION!
ANY PRODUCTION KEYS SHOULD BE GENERATED BY THE ORGANIZATION AND BE 4096 BITS LONG.

Sample device certificate

The sample device x.509 certificate, signed by the root certificate above, was generated with the following command:

```
# Create the device key. In production, increase the bit size to 4096+
openssl genrsa -out insecure_device_key.pem 1024

# Create a certificate signing request
openssl req -new -key insecure_device_key.pem -out insecure_device_cert.csr

# Sign the signing request and create the insecure_device_cert.crt file
openssl x509 -req -days 365 -in insecure_device_cert.csr \
             -CA insecure_root_cert.crt -CAkey insecure_root_key.pem \
             -set_serial 01 -out insecure_device_cert.crt
```

The `insecure_device_cert.csr` intermediate certificate signing request can be deleted once the three steps above have been executed.

The content of the sample `insecure_device_key.pem` private key file, which has no password protection, is:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCbb4jPAESKxarj3NJsgfQbhfTHZAP9kmram2TFnkzlCRxq4wQx
BDC0O85PAMgZou0armGGbOu0si4cpVRioerCQJXnMWx1MI+3GUktW5ijI3ui+tYC
sMQZtjSBVNXFZdoyZU2lPVWITOMZOe8o9vJ5DcUmFj9b2xV9jQ19oh+2+QIDAQAB
AoGAVCYV0rs6YEaTNbke0k+ocB4dXrTu1CCoaKEn9TS2PGiqUdOFOWQjWe/myS6L
JhXmd0Ng2P2uvayY+jknbh5qkNeEgTDhXJlAjiXlCADYArhgib+evRHgKz7RLTjX
tGklbmc7oECTEpjkchJC5XcJhXzHCIjroyOJvBuAVa+SeAECQQDNC+KW7fTKQpiG
YNGIt5MxCMjRparLz0fWod9J9U56wrWzU9Rnb7h9iwzTEJUEcVl9z8rnUdWtYQ8X
3lsz5cDhAkEAwg+kDWbLtXWlIvXhhla7q0+RfKb8vu/gXnkXJa6rcJdJztKRbP3b
9fehVeu9m+1+abahjC1zmQimwd2QVc8BGQJADbtfCGaVPzpoho9TWQmaRO1mrYuf
vZh7IiejEYvpHpWNn53cmrTDsTyvti7lG/APYzqYRxeW7M6UOS/+AaLAYQJAJbEW
AwhZPphoB59MO2RzNPXSYyyn4IoEwTSxuz7uy4KG8mXRmyK/a0m6i06rWDLLn8q6
G9jkH/AfO35GP3RiWQJBAJLWBlKpHf8TxT65jAwxBhd9ZOkC2w0WidbSYjX9wkkD
38K7ZDm1LSIR69Ut6tdwotkytXvDniOMPY6ENar5IUs=
-----END RSA PRIVATE KEY-----
```

The content of the sample `insecure_device_cert.crt` file is:

```
-----BEGIN CERTIFICATE-----
MIIB8zCCAVwCAQEwDQYJKoZIhvcNAQELBQAwQjELMAkGA1UEBhMCVVMxDzANBgNV
BAgMBk9yZWdvbjERMA8GA1UEBwwIUG9ydGxhbmQxDzANBgNVBAoMBkdFTklWSTAe
Fw0xNTExMjcyMzE0NTJaFw0xNjExMjYyMzE0NTJaMEIxCzAJBgNVBAYTAlVTMQ8w
DQYDVQQIDAZPcmVnb24xETAPBgNVBAcMCFBvcnRsYW5kMQ8wDQYDVQQKDAZHRU5J
VkkwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAJtviM8ARIrFquPc0myB9BuF
9MdkA/2SatqbZMWeTOUJHGrjBDEEMLQ7zk8AyBmi7RquYYZs67SyLhylVGKh6sJA
lecxbHUwj7cZSS1bmKMje6L61gKwxBm2NIFU1cVl2jJlTaU9VYhM4xk57yj28nkN
xSYWP1vbFX2NDX2iH7b5AgMBAAEwDQYJKoZIhvcNAQELBQADgYEAhbqVr9E/0M72
9nc6DI+qgqsRSMfoyvA3Cmn/ECxl1ybGkuzO7sB8fGjgMQ9zzcb6q1uP3wGjPioq
MymiYYjUmCTvzdvRBZ+6SDjrZfwUuYexiKqI9AP6XKaHlAL14+rK+6HN4uIkZcIz
PwSMHih1bsTRpyY5Z3CUDcDJkYtVbYs=
-----END CERTIFICATE-----
```

These files are checked into `priv/certificates` and `priv/keys`.

DO NOT USE THE KEYS AND CERTIFICATES ABOVE IN PRODUCTION!
ANY PRODUCTION KEYS SHOULD BE GENERATED BY THE ORGANIZATION AND BE 4096 BITS LONG.

RVI credentials format

A credential is a JWT-encoded JSON structure, signed by the root X.509 certificate's private key, describing the rights that the sender has. A received RVI credential is validated as follows.

1. **Receive remote party's X.509 device certificate**
   The TLS handshake process will exchange the X.509 certificates setup in the previous chapter.

2. **Validate remote party's X.509 device certificate**
   The received device X.509 certificate has its signature validated by the root X.509 certificate that is pre-provisioned in all RVI nodes.
   The receiver now knows that the remote RVI node has an identiy generated by a trusted provsioning server using the private root key.

3. **Receive one or more RVI credentials**
   Each credential is encoded as JWT, signed by the root X.509 certificate.

4. **Validate each RVI credential signature**
   The root X.509 certificate is used to validate the signature of each received RVI credential.
   A successful validation proves that the certificate was generated by a trusted provisioning server using the private root key.

5. **Validate the credential-embedded X.509 device certificate**
   Each received RVI credential will have its embedded device X.509 certificate compared with the device X.509 certificate received in step 1 above.
   A match proves that the certificate was generated by a trusted provisioning server explictly for the RVI node at the remote end.

An RVI credential has the following format in its native JSON state:

```
{
    "create_timestamp": 1439925416,
    "right_to_invoke": [
        "jlr.com/vin/"
    ],
    "right_to_register": [
        "jlr.com/backend/sota"
    ],
    "id": "insecure_cert",
    "iss": "jaguarlandrover.com",
    "device_cert": "",
    "validity": {
        "start": 1420099200,
        "stop": 1925020799
    }
}
```

The members are as follows:

Member	Description
create_timestamp	Unix timestamp of when the credential was created
right_to_invoke	A list of service prefixes that the sender has the right to invoke on any node that has registered matching services that start with the given string(s).
right_to_register	A list of services that the sender has the right to to register for other nodes to invoke.
id	A system-wide unique identifier for the credential.
iss	The issuing organization.
device_certificate	The PEM-encoded device X.509 certificate to match against the sender's TLS certificate.
validity.start	The Unix timestamps when the credential becomes active.
validity.stop	The Unix timestamps when the credential becomes inactive.

Generating RVI credentials

To create a credential, tie it to a device X.509 certificate, and sign it with a root X.509 certificate private key, the following command is used:

```
rvi_create_credential.py --cred_out="insecure_credential.json" \
                         --jwt_out='insecure_credential.jwt' \
                         --id="xxx" \
                         --issuer="genivi.org" \
                         --root_key=insecure_root_key.pem \
                         --device_cert=insecure_device_cert.crt \
                         --invoke='genivi.org/' \
                         --register='genivi.org/'
```

The following command line parameters are accepted:

Parameter	Required	Description
--cred_out	No	Output file containing the JSON-formatted un-encoded credential.
--jwt_out	Yes	JWT-encoded, JSON-formatted, root keyp-signed credential.
--issuer	Yes	Organization that issued the credential.
--root_key	Yes	Private, PEM-encoded root key to sign the credential. Must be the same key used to sign the root X.509 certificate.
--device_cert	Yes	The PEM-encoded device X.509 certificate to embed into the credential as the device_cert member.
--invoke	Yes	Space separated list (within quotes) of RVI service prefixes that the owner of the credential has the right to invoke.
--register	Yes	Space separated list (within quotes) of RVI service prefixes that the owner of the credential has the right to register for others to call (with the right credential).
--start	No	The Unix timestamps when the credential becomes active.
--stop	No	The Unix timestamps when the credential becomes inactive.

The generated `insecure_credential.json` and `insecure_credential.jwt` are checked into `priv/credentials`.