OTrP proto

National Institute of Advanced Industrial Science and Technology

2019-09-10

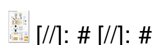
1 OTrP Overview 1
1.1 Normative docs . . . 1
1.2 Working overview diagram . . . 1
1.3 TA Encryption . . . 3
1.4 OTrP PKI . . . 4
1.5 OTrP sources layout . . . 4
1.6 OTrP test flow . . . 4
1.6.1 Install Test TA flow . . . 5
1.6.2 Confirm Test TA is installed . . . 5
1.6.3 Delete Test TA flow . . . 6
1.6.4 Confirm Test TA deleted . . . 6
2 Notes on PKI config 6
2.1 One-time PKI setup . . . 6
2.2 PKI layout . . . 6
2.3 Converting and using JWK/E/S manually . . . 7
2.3.1 Step 1: build libwebsockets with mbedtls . . . 7
2.3.2 Step 2: run the pki kickstart script . . . 8
2.3.3 Step 3: run the test script . . . 8

1 OTrP Overview

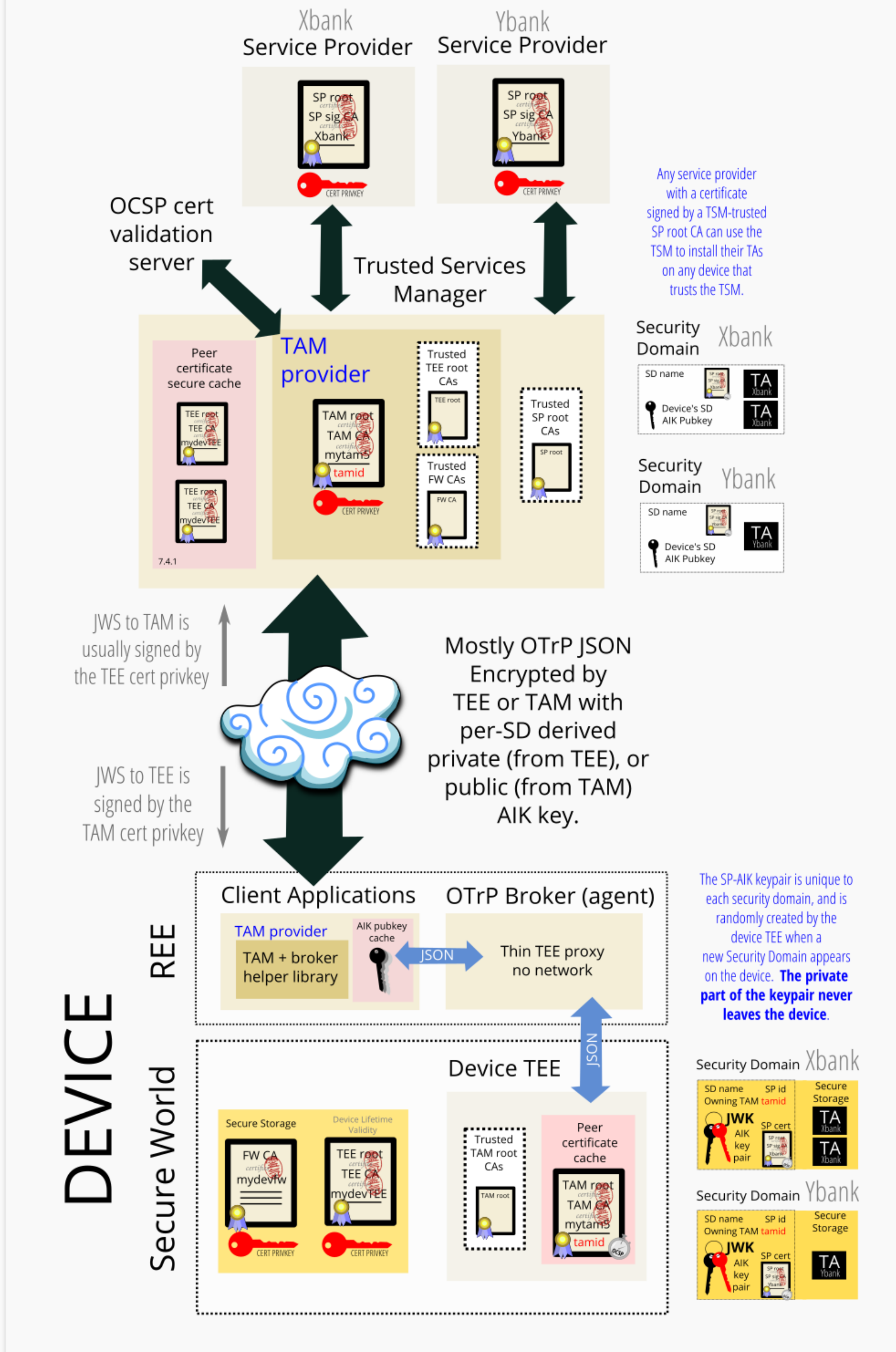
1.1 Normative docs

- OTrP draft RFC
- globalplatform GPD TMF OTrP profile (public review v0.0.0.21)
- JWS RFC7515
- JWE RFC7516
- JWK RFC7517
- JWA RFC7518

1.2 Working overview diagram

 [//]: # [//]: #

Copyright © National Institute of Advanced Industrial Science and Technology (AIST)



Copyright © National Institute of Advanced Industrial Science and Technology (AIST)

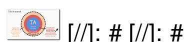
OTrP proto provides additional features to enhance existing GlobalPlatform based TEE. It is developed on top of OP-TEE.

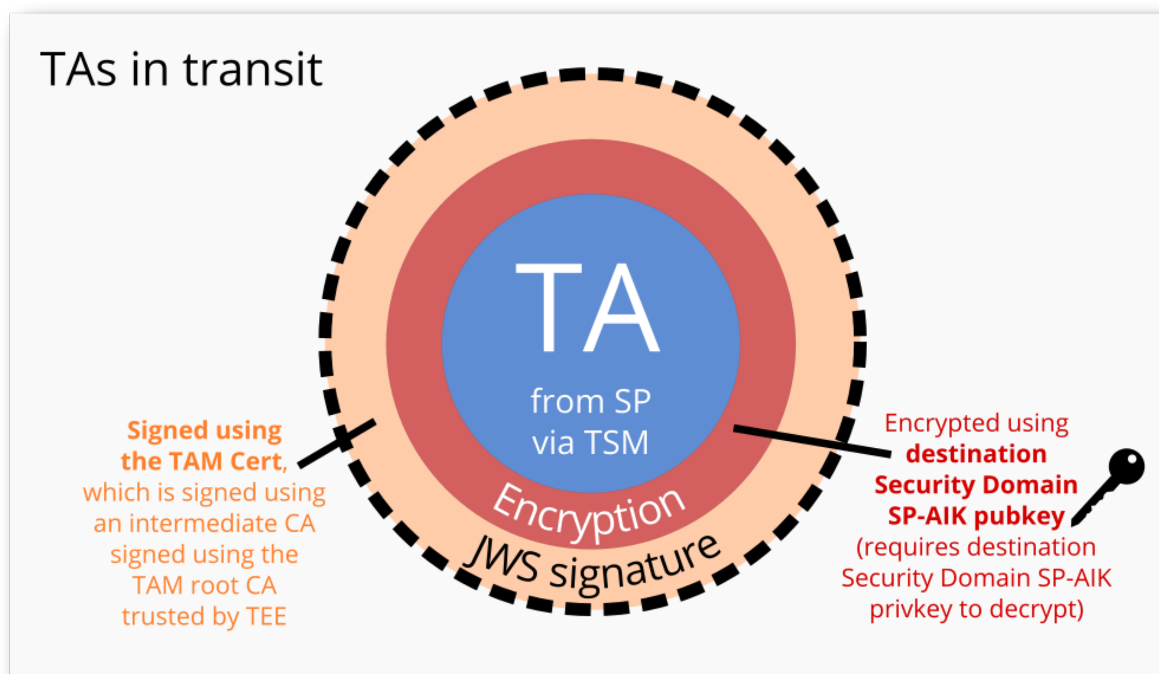
1. Capability to have encrypted TAs which could contain security sensitive binaries and data.
2. Enabling TAs being signed by different TA providers, not only one valid signing key defined at TEE build-time.

Details of achieving the two features.

- To solve TA encryption, they are encrypted by a remote provider (the "Trusted Services Manager" or TSM) using a public key held by the targeted device specifically, before transmission to the device. They may be stored with the encryption intact and the decrypted and confirmed on insertion.
- So one provider can't snoop the TAs of another, there are separate keypairs derived on the device per provider (held in a Security Domaiin (SD) per- "Service Provider" (SP)
- A new TA insertion method is implied on the TEE-side that decrypts "installed" TAs and inserts them after confirming decryption, without checking for the old build-time trusted signature.
- A granular PKI is defined using X.509 certs that allows the remote Trusted Services Manager (TSM) to confirm if it is talking to something with access a TEE certificate on the other side. The TEE side holds a list of TSM certifcates it is willing to trust and can confirm the packets it is receiving came from something with access to one of those.
- The client application initiates most activities, and includes a library that deals with network connectivity. However almost all of the traffic passing through the client application and the OTrP broker has a payload encrypted with keys unavailable to either the client application or the OTrP broker. The Trusted Services Manager (TAM) and the TEE side are the two endpoints for the encrypted communication that have the necessary keys to see plaintext.
- Network-traversing packets are placed in a "Flattened JSON" JWS signed wrapper. JWS payload that fails the signature check is discarded without being processed, making it difficult to trigger bugs or corner cases by fuzzing type attacks or targeted hacks like buffer overflows.

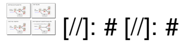
1.3 TA Encryption

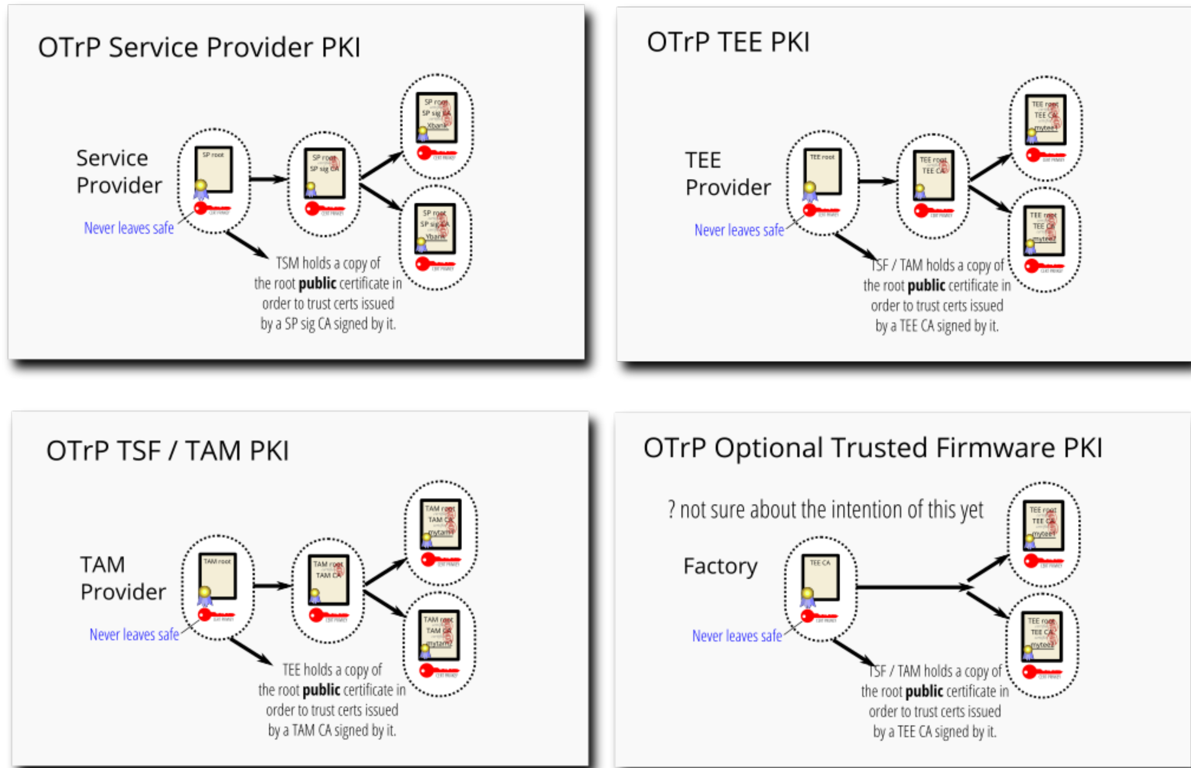
 [//]: # [//]: #



Copyright © National Institute of Advanced Industrial Science and Technology (AIST)

1.4 OTrP PKI

 [//]: # [//]: #



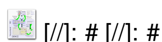
Step 1: Run otrp-kickstart-pki.sh as described below

Setting up OTrP PKI

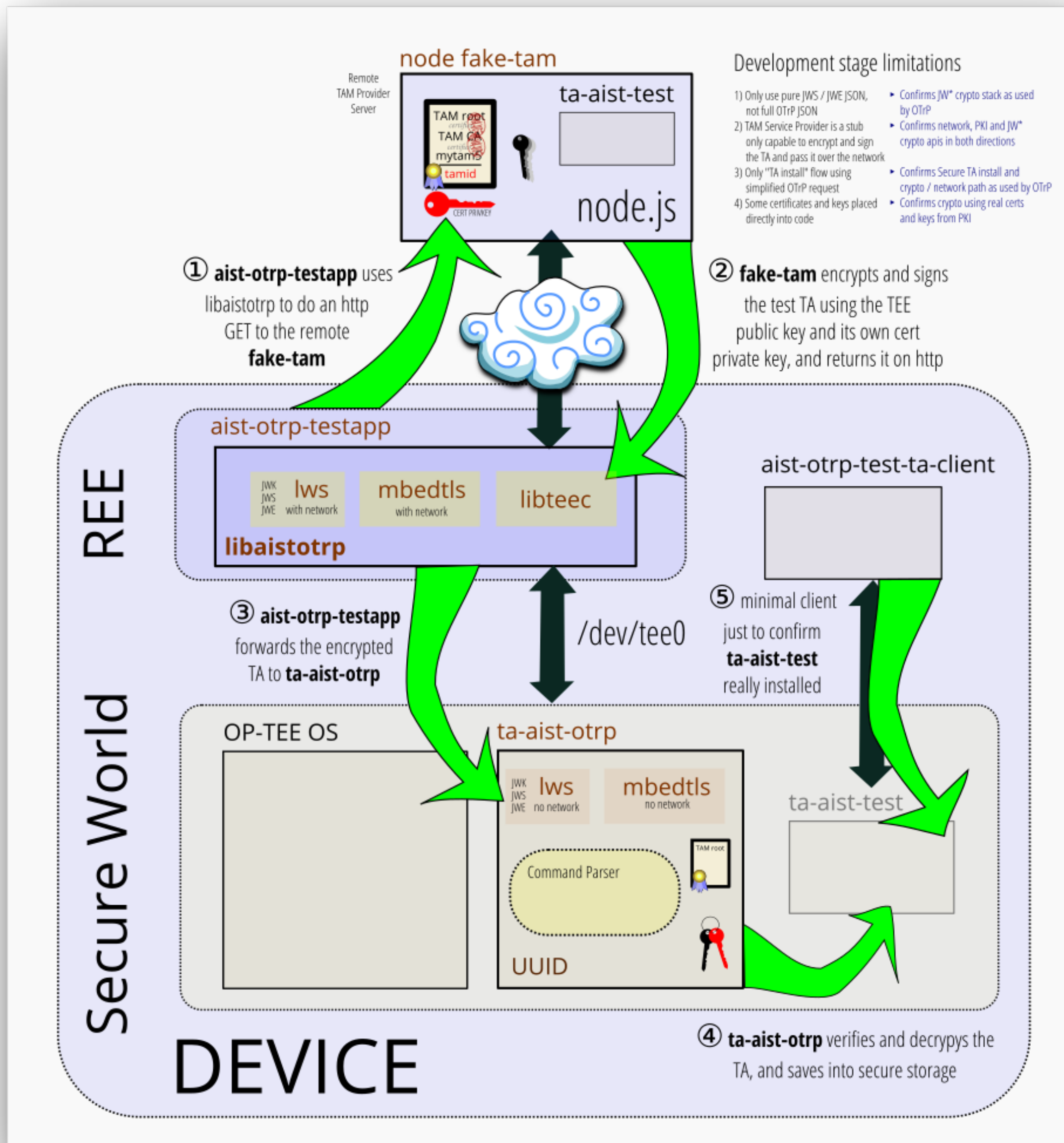
1.5 OTrP sources layout

Subdirectory	Function
teep-broker-app	Test REE client application, uses libteep to fetch an encrypted test TA from TAM and install it
sp-hello-app	Tiny REE client that just opens a session to the test TA if it is installed successfully
libteep	REE shared library that can do http(s) requests to the TAM and can forward results to teep-agent-ta
pki	PKI created by scripts/otrp-kickstart-pki.sh
teep-agent-ta	TA implementing OTrP on TEE side
sp-hello-ta	Tiny TA that is copied to the fake TAM so it can be encrypted and sent to the TEE via teep-broker-app

1.6 OTrP test flow

 [//]: # [//]: #

Copyright © National Institute of Advanced Industrial Science and Technology (AIST)



Start the fake TAM on the remote server

```
# node ./tiny-tam/app.js
```

1.6.1 Install Test TA flow

```
# ifconfig eth0 192.168.2.22/24 up
# echo "nameserver 192.168.2.1" >  /etc/resolv.conf
# echo "192.168.2.236 buddy.home.your-server" > /etc/hosts
# teep-broker-app --tamurl http://buddy.home.your-server:3000
```

1.6.2 Confirm Test TA is installed

```
# sp-hello-app
START: sp-hello-ap
I/TA: TA_InvokeCommandEntryPoint:
I/TA: sp-hello-ta: Hello IETF TEEP!
sp_hello_app: done
```

Copyright © National Institute of Advanced Industrial Science and Technology (AIST)

1.6.3 Delete Test TA flow

```
# teep-broker-app --tamurl http://buddy.home.your-server:3000 -d
```

1.6.4 Confirm Test TA deleted

```
# sp-hello-app
START: sp-hello-app
ERR [649] TEES:load_ta:225:   TA not found
E/TC:? 0 tee_ta_open_session:540 init session failed 0xffff0008
Could not open session with TA
```

2 Notes on PKI config

2.1 One-time PKI setup

You can configure the root certs, intermediate CA certs and some selected demo certs signed by the intermediate CA certs by running this script.

It also converts some PEM certs and keys to JWK for later tests (and hence needs the path to the libwebsockets conversion tools)

```
$ LWS_BUILD_DIR=/path/to/libwebsockets/build ./scripts/otrp-kickstart-pki.sh
```

2.2 PKI layout

There are three cerificate chains laid out the same way, for "Service Provider" (SP), "Trusted Application Manager" (TAM) and "Trusted Execution Environment" (TEE).

For SP as an example:

```
./pki
 sp
  sp-rootca
   sp-rootca-ec-key.pem
   sp-rootca-ec-cert.pem
  sp-ca
   sp-ca-1-ec-key.pem
   sp-ca-1-ec-cert.pem
  sp
   sp-ybank-rsa-key.pem
   sp-ybank-rsa-cert.pem
   sp-ybank-rsa-cert-plus-intermediate-cert.pem
   sp-xbank-rsa-cert.pem
   sp-xbank-rsa-key.pem
   sp-xbank-rsa-cert-plus-intermediate-cert.pem
```

The pki for TAM (./pki/tam...) and TEE (./pki/tee/...) is the same except signed certificates are produced for "mytam" and "mytee" instead of the fictional bank SPs.

Root keys are produced for both rsa and ec, and which to use can be selected when creating the intermediate CAs. The kickstart script uses ec keys for the rootca to sign the intermediate CA, and 4096-bit RSA keys for the intermediate CA to sign the final certificates.

For the final signed certs, the normal certificate with the public key is produced, but also a `...-cert-plus-intermediate-cert.pem` bundle that contains both the normal signed cert and a copy of the intermediate CA cert.

Copyright © National Institute of Advanced Industrial Science and Technology (AIST)

2.3 Converting and using JWK/E/S manually

A test script is provided which uses the PKI infrastructure to encrypt, sign and ”send” a ”TA” from the ”TAM side” to the ”TEE side”, where it's verified for signature and decrypted and confirmed to be unchanged from the original.

This flow has important deviations from OTrP...

- it runs on a PC not the TEE and TAM server
- it doesn't do any communication
- the JWE and JWS are stock RFC without OTrP extensions, it doesn't attempt to make correctly formed OTrP request or response packets just standard JWE / JWS
- it deals with 50KiB of random instead of a real TA and doesn't try to use the TA in a TEE
- there are no test vectors available, so it decrypts its own encryption and verifies its own signatures. So there are no guarantees the crypto flow is interoperable yet.

... however...

- it's using exactly the same libwebsockets JW∗ code as the TEE
- it's using exactly the same generic crypto code as the TEE
- it's using exactly the same mbedtls crypto backend as the TEE
- it's using real JWS and JWS crypto agility alg / enc
- it's using the actual ECDH and RSA / AES crypto as used by OTrP
- it's using the actual PKI infrastructure with the actual EC and RSA root CAs, intermediate CAs and certs for TAM and TEE
- it's using real cert keys converted to JWK on the fly for the correct crypto flow

... so it proves a small but significant part of one flow around the certs and crypto.

2.3.1 Step 1: build libwebsockets with mbedtls

Build libwebsockets on your build machine and to build the minimal examples.

You should install your either your distro mbedtls first (-DLWS_WITH_MBEDTLS=1)

Distro	Package
Fedora	mbedtls-devel
Ubuntu	libmbedtls-dev

or distro OpenSSL (-DLWS_WITH_MBEDTLS=0)

Distro	Package
Fedora	openssl-devel
Ubuntu	libssl-dev

Copyright © National Institute of Advanced Industrial Science and Technology (AIST)

```
$ git clone https://libwebsockets.org/repo/libwebsockets
$ cd libwebsockets
$ mkdir build
$ cd build
$ cmake .. -DLWS_WITH_JOSE=1 -DLWS_WITH_MBEDTLS=1 -DLWS_WITH_MINIMAL_EXAMPLES=1
$ make -j12 && sudo make install
$ sudo ldconfig
```

This will build but not install several dozen example applications in ./build/bin, which are able to perform operations from the commandline on x.509 PEM, JWKs, JWS (signing) and JWE (en/decrypt). These can be used to synthesize the core operations around, eg, encrypting and signing a TA and verifying and decrypting it, using the PKI created above.

2.3.2 Step 2: run the pki kickstart script

```
$ LWS_BUILD_DIR=/path/to/libwebsockets/build ./scripts/otrp-kickstart-pki.sh
```

This recreates all the PKI, root certs and keys etc... running this again will mean you will have to rebuild everything and update tiny-tam copies of the crypto etc using the new pki.

2.3.3 Step 3: run the test script

Run the test script like this

```
$ LWS_BUILD_DIR=/path/to/libwebsockets/build ./scripts/otrp-test.sh
```

You should see output like this

```
[2019/01/03 19:13:10:0065] USER: LWS X509 api example
[2019/01/03 19:13:10:0067] NOTICE: lws_x509_public_to_jwk: RSA key
[2019/01/03 19:13:10:0123] NOTICE: Issuing Cert + Private JWK on stdout
[2019/01/03 19:13:10:0123] NOTICE: main: OK
[2019/01/03 19:13:10:0135] USER: LWS X509 api example
[2019/01/03 19:13:10:0137] NOTICE: lws_x509_public_to_jwk: RSA key
[2019/01/03 19:13:10:0137] NOTICE: Issuing Cert Public JWK on stdout
[2019/01/03 19:13:10:0138] NOTICE: main: OK
[2019/01/03 19:13:10:0243] USER: LWS JWK example
[2019/01/03 19:13:10:0244] NOTICE: lws_jwk_generate: generating 4096 bit RSA key
[2019/01/03 19:13:10:6649] USER: LWS JWE example tool
[2019/01/03 19:13:10:6703] USER: LWS JWS example tool
[2019/01/03 19:13:10:6942] USER: LWS JWS example tool
[2019/01/03 19:13:10:6989] NOTICE: VALID
[2019/01/03 19:13:10:7002] USER: LWS JWE example tool
Decrypted TA matches original
```

Running `otrp-test.sh` doesn't change the pki, so you can run it as many times as you like.

Copyright © National Institute of Advanced Industrial Science and Technology (AIST)