Testing an HSM for OpenDNSSEC

As far as OpenDNSSEC is concerned, any PKCS #11 implementation can be used as its underlying HSM. This means that an important part of HSM testing is PKCS #11 testing. However, as the SoftHSM is a separate deliverable that happens to emanate from the OpenDNSSEC project, we do include a few specific tests for the SoftHSM.

HSM and SoftHSM

Existing test suites

One test suite is available on http://www.mozilla.org/projects/security/pki/pkcs11/netscape/ and is described as follows: The Netscape PKCS #11 Test Suites are designed to help vendors of PKCS #11-compliant cryptographic hardware verify compatibility with Netscape software.

The HSM must pass this test.

Requirements

General remarks

All tests that follow should be run with tests for memory leaks, and the free memory available inside the SoftHSM should be returned to the same amount as before the test.

Preferrably, these tests should not cause fragmentation. If fragmentation does occur, the test should be repeated until no *more* fragmentation occurs. This should happen within three test runs.

All tests assume an initialised token. If information is required to be present on the token, it will be created and destroyed as part of the test.

The distinction between dynamic and static linking of a library is not taken into account in what follows; this is because it is not practical to unload and later reload a library.

Initiation sequence

The PKCS #11 setup sequence comprises of a number of steps. The HSM should support any partial setup with the mirrorring breakdown:

- FS find an available slot to access
- S+ open a session with that slot
- A+ login to access the token in the slot (variations: RO User, RW User, RW SO)

- TS attempt operations for each of the login variations, and check if their success or failure match the specifications for the login variations
- A- logout from the token
- S- close the session with that slot

Sequences to test include:

1. FS S+ A+(ROUSER) TS A- S-
2. FS S+ A+(RWUSER) TS A- S-
3. FS S+ A+(RWSO) TS A- S-

Also, arbitrary steps marked with a + in these sequences should at random, but occassionally (say, 10% chance for each step, so the chance of a proper sequence is over 70%) be skipped. If a + step is skipped it must cause errors in the following steps, at least in the TS steps and the - matching the missing + step.

If handles are needed but not available from previous in-sequence function calls, use values from earlier attempts, or 0 if none were available. Only those inherited from an earlier FS in a sequence without it may lead to successes. Note that this implies that unstructured sequences are tried, like:

- FS A+(RWSO) TS A- S-

Numerous of these sequences should be run in random order (say, 100 in a sequence) and the last one broken off halfway. This should leave the token in a well-defined and as-expected state. If an object was being created during the break-off, then either it must be present in its entirety or it should not be present at all -- the basic property of atomicity.

Memory fragmentation test

OpenDNSSEC will create and destroy large numbers of keys. This behaviour must not lead to clogged up HSMs.

1. Create as many keys, in various sizes, as will fit onto the token. If a key cannot be created it should fail due to memory limits, and smaller key sizes can be tried to fill up the space until no more will fit into the HSM.
2. Pick a key at random, remove it, and create a new one of the same size. This test must not fail, as this would indicate memory fragmentation.
3. Repeat the previous step; it should run thousands (say, 2500) of times.
4. Destroy all keys.

It is possible to extend this into a linearity test of the memory space:

- Perform the test above multiple (say, 10) times and each time create the same key sizes as in step 1, but in a different order. After having created all the keys that were successful in step 1, try once more with all the key sizes that failed and thereby ended step 1. Repeat the rest of the procedure as stated above, possibly with less repeats of step 2 (say, 250 times) and expect the same results.

Throughout this process, any monitoring of memory performance in the HSM is welcome. Minimally, a memory leak test should be performed to ensure that the consumption of memory in the implementation of the HSM's memory space does not leak. Any tests that monitor fragmentation may be of use to see if the HSM would start to underperform at some point.

Key creation test

This test ensures that RSA keys of the required sizes are available.

1. Start off with the set of key sizes that are demanded by OpenDNSSEC
2. Inquire what other sizes are supported by the HSM. If ranges are supported, add every occurring size but keep in mind that an RSA modulus always has a length which is a multiple of 8.
3. Create an RSA key pair with that length.
4. Export the key's modulus.
5. Ensure that the length matches the specifications.
6. Destroy the key pair.
7. Iterate from 3 until all key sizes have been tested.

Signing test

1. Create an RSA key pair of a size demanded by OpenDNSSEC (pick at random)
2. Create a lot of signatures with it, signing random data (say, 100 signatures)
3. Verify each of these signatures against the public key
4. Destroy the RSA key pair
5. Repeat from step 1 for a couple of times (say, 10)

Reliable state recovery test

Any HSM must be a solid, reliable tool. It should be able to withstand sudden disruptions in the supplied power, and come back up gracefully. In all cases, the data that has been confirmed to exist must persist after any form of crash. Also, crashes may not lead to the unavailability of an HSM resource.

HSM Toolkit

The code for the HSM Toolkit is currently subjected to major work. We have not established a test plan for them. In general, the HSM Toolkit may help to perform the foregoing tests, but it may make sense to test it on its own.

To be able to perform any tests, documentation with the intended use of the software will be required.