<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Connector Reference</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>HTTPS Connections</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Configuration</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>YubiHSM Shell Reference</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>How to Use the Shell</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>Data Format</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>Enabling Debug</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>YubiHSM 2 Setup Tool Reference</td>
<td>5</td>
</tr>
<tr>
<td>3.1</td>
<td>Setup for EJBCA</td>
<td>5</td>
</tr>
<tr>
<td>3.2</td>
<td>How It Works</td>
<td>6</td>
</tr>
<tr>
<td>3.3</td>
<td>Backing Up the YubiHSM 2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Libyubihsm Reference</td>
<td>7</td>
</tr>
<tr>
<td>4.1</td>
<td>Backends</td>
<td>7</td>
</tr>
<tr>
<td>4.2</td>
<td>HTTP Connector</td>
<td>7</td>
</tr>
<tr>
<td>4.3</td>
<td>USB Connector</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Python Library Reference</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>YubiHsm Wrap Reference</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>Key Storage Provider Reference</td>
<td>13</td>
</tr>
<tr>
<td>7.1</td>
<td>Additional Documentation for YubiHSM Key Storage Provider</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>Creating a Code-Signing Certificate using the Key Storage Provider</td>
<td>15</td>
</tr>
<tr>
<td>8.1</td>
<td>Configure the Key Storage Provider</td>
<td>15</td>
</tr>
<tr>
<td>8.2</td>
<td>Create the Certificate Request Configuration File</td>
<td>16</td>
</tr>
<tr>
<td>8.3</td>
<td>Create the Certificate Request</td>
<td>17</td>
</tr>
<tr>
<td>8.4</td>
<td>Sign the Certificate Request</td>
<td>17</td>
</tr>
<tr>
<td>8.5</td>
<td>Sign using Signtool</td>
<td>17</td>
</tr>
<tr>
<td>8.6</td>
<td>Troubleshooting</td>
<td>17</td>
</tr>
<tr>
<td>8.7</td>
<td>More Information</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>Move Software Keys to Key Storage Provider</td>
<td>19</td>
</tr>
<tr>
<td>9.1</td>
<td>Export your Existing Private Key and Certificate</td>
<td>19</td>
</tr>
<tr>
<td>9.2</td>
<td>Import the Target Private Key</td>
<td>20</td>
</tr>
<tr>
<td>9.3</td>
<td>Restore the Target Certificate</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>Status Codes Reference</td>
<td>23</td>
</tr>
</tbody>
</table>
The \textit{yubihsm-connector} performs the communication between the YubiHSM 2 and the applications that use it.

The Connector must have permissions to access the USB device, and different operating systems behave differently in this regard. The easiest way to get started is to run the Connector with Administrator privileges (e.g. with \textit{sudo}), but the safest way to run the Connector is to use your operating system’s configuration to give it only the privileges necessary to access the YubiHSM 2 USB device.

The Connector is not a trusted component. Sessions are established cryptographically between the application and the YubiHSM 2 using a symmetric mutual authentication scheme that is both encrypted and authenticated.

The Connector is not required to run on the same host as the applications which access it. In that case the Connector should be configured to be listening on a different address and port rather than the default \texttt{localhost:12345}, making sure that the client has access.

The protocol is tunneled over HTTP, but it is not a RESTful API or similar. It is however possible to get information regarding the Connector by issuing a GET request on the \texttt{/connector/status} URI.

### 1.1 HTTPS Connections

As mentioned earlier, the Connector is not meant to be a trusted component. For this reason it defaults to HTTP connections. It is possible to use HTTPS, however this requires providing a key and a certificate to the Connector.

Another option is to use a reverse proxy such as nginx before the Connector and have that handle TLS.

### 1.2 Configuration

Sample configuration for the Connector: \texttt{yubihsm-connector-config.yaml}

```
# Certificate (X.509)
cert: ""

# Certificate key
key: ""

# Listening address. Defaults to "localhost:12345".
listen: localhost:12345

# Device serial in case of multiple devices
serial: ""
```

(continues on next page)
# Log to syslog/eventlog. Defaults to "false".
syslog: false

# Use to enable host header filtering. Default to "false".
# Use this if there is an absolute need to use a web browser on the
# host where the YubiHSM 2 is installed to connect to untrusted web
# sites on the Internet.
enable-host-whitelist: false

# Default list for the host header filter
host-whitelist: localhost, localhost., 127.0.0.1, [::1]

Sample udev rule to be placed into /etc/udev/rules.d/:

# This udev file should be used with udev 188 and newer
ACTION="add|change", GOTO="yubihsm2_connector_end"

# YubiYubiHSM 2
# The OWNER attribute here has to match the uid of the process
# running the Connector
SUBSYSTEM="usb", ATTRS{idVendor}"1050", ATTRS{idProduct}"0030",
    OWNER="yubihsm-connector"
LABEL="yubihsm2_connector_end"
CHAPTER TWO

YUBIHSM SHELL REFERENCE

The yubihs-shell is the administrative and testing tool you can use to interact with and configure the YubiHSM 2 device.

The Shell can be invoked in two different ways: interactively, or as a command line tool useful for scripting.

Additional information on the various commands can be obtained with the help command in interactive mode or by referring to the --help argument for the command line mode.

Examples of commands can also be found in the Command reference.

Note: For signing, releases prior to and including the current yubihsm2-sdk release have an 8192-byte file size limit - 4k in interactive mode, or 8k in non-interactive mode.

2.1 How to Use the Shell

2.1.1 Command Syntax

Commands and subcommands require specific arguments to work. The Shell will return an error message if the command syntax is incorrect, pointing at the first invalid argument.

Arguments have different types. In interactive mode pre-defined values for command types can be tab-completed (Tab Completion does not work on Windows).
2.1.2 Possible Command Types

<table>
<thead>
<tr>
<th>Arg</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>number</td>
<td>A generic (hex or dec) unsigned number</td>
</tr>
<tr>
<td>w</td>
<td>word</td>
<td>A generic (hex or dec) 16-bit unsigned number</td>
</tr>
<tr>
<td>b</td>
<td>byte</td>
<td>A generic (hex or dec) 8-bit unsigned number</td>
</tr>
<tr>
<td>i</td>
<td>input data</td>
<td>Input data, generally defaults to standard input</td>
</tr>
<tr>
<td>F</td>
<td>output filename</td>
<td>Output file name, generally defaults to standard output</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
<td>A generic string (use quotes for strings including white spaces)</td>
</tr>
<tr>
<td>e</td>
<td>session</td>
<td>The ID of an already-established Session</td>
</tr>
<tr>
<td>d</td>
<td>domains</td>
<td>A list of Domains, either in hex (ex: 0xffff) or string form (ex: 3,5,14)</td>
</tr>
<tr>
<td>c</td>
<td>capabilities</td>
<td>A list of Capabilities in either form: hex (ex: 0xffffffffffffffff) or string (ex: sign-pkcs,sign-pss, get-log-entries)</td>
</tr>
<tr>
<td>a</td>
<td>algorithm</td>
<td>An algorithm in string form (ex: eccp256)</td>
</tr>
<tr>
<td>t</td>
<td>type</td>
<td>An Object Type in string form (ex: Asymmetric)</td>
</tr>
<tr>
<td>o</td>
<td>option</td>
<td>A device-global option in string form (ex: force-audit)</td>
</tr>
<tr>
<td>I</td>
<td>format</td>
<td>A format specifier in string form (ex: base64)</td>
</tr>
</tbody>
</table>

2.2 Data Format

Different commands have different default formats. These can be listed by invoking help on a specific command. For example, the help sign will contain the following lines:

```
pss   Sign data using RSASSA-PSS (default input format: binary)
       e:session,w:key_id,a:algorithm,i:data=-,F:out=-
```

As can be seen, the input format is binary. Additionally, arguments to a command that have =- after their type and name (like i:data and F:out in the example above), use the standard input or standard output by default for reading data.

2.3 Enabling Debug

Different levels of debug output can be enabled by using the -v flag in command line mode, or by issuing the debug LEVEL command in interactive mode, where LEVEL is one of all, crypto, error, info, intermediate, none or raw.
The SDK ships with a tool called yubihsm-setup that helps with setting up a device for specific use cases. The tool assumes familiarity with the key concepts of YubiHSM such as Domains, Capabilities and Object IDs. It currently supports the following:

- setup for KSP/ADCS and EJBCA;
- restoring a previous configuration
- resetting the device to factory defaults
- exporting all existing objects

The tool is based around the concept of secret-sharing. When setting up Objects, those are exported with a freshly created Wrap Key. The key is never stored on disk, but rather it is printed on the screen as shares. The key concepts here are:

- The number of shares, which is the number of parts the key should be divided into
- The security threshold, which is the minimum number of shares required to reconstruct the Wrap Key.

Besides splitting the Wrap Key into shares, the tool (by default) also exports under wrap all the newly created objects and saves them in the current directory. This can be used at a later time to “clone” or recover a device. This operation can be performed either with yubihsm-setup or manually if the Wrap Key is known.

By default, the Authentication Key used to establish a Session with the device is also normally deleted at the end of the process.

Default behavior can be altered with command line options. For more information, consult the tool’s help.

3.1 Setup for EJBCA

When setting up the device for use by EJBCA, the setup tool will also generate an asymmetric keypair and an X509 certificate suitable for use as a CA key. The setup tool can be re-run as many times as the number of asymmetric keys to be generated since each run will produce only one keypair and one corresponding X509 certificate.

Note: Using the --no-new-authkey flag will prevent generation of a new Wrap Key and a new Authentication Key.
3.2 How It Works

For the JAVA implementation, a keypair can be used to perform PKCS#11 operations only if the key and its corresponding X509 certificate are stored under the same ID on the device (the value of their CKA_ID attributes is the same). To store them under the same ID, run the YubiHSM 2 Setup tool with the ejbca subcommand:

1. Generate an Asymmetric Key on the YubiHSM 2.
2. Generate an attestation certificate for the asymmetric key and import it into the YubiHSM 2 under the same ID as the Asymmetric Key.

The attestation certificate stored on the YubiHSM 2 is, in fact, only a placeholder certificate for the public key. It is never used by EJBCA because EJBCA stores the CAs’ certificates in a dedicated database.

3.3 Backing Up the YubiHSM 2

To back up the entire content of the device, we recommend you use the dump subcommand on one YubiHSM and then the restore subcommand on another YubiHSM. For more information, consult the Setup tool’s online help.
Libyubihsm is the C library used to communicate natively with a YubiHSM 2. It implements and exposes convenience functions for all the commands supported by the device. It also allows the sending of unformatted “raw” messages over an established session or in plain text.

The library is used by:

- **yubihs-shell**, see *YubiHSM Shell Reference*
- **PKCS#11 module**, see *PKCS#11 with YubiHSM 2 Reference*
- **KSP**, see *Key Storage Provider Reference*

Documentation of the library API can be found as comments within the header file (yubihsm.h) in the SDK, or as a pre-built Doxygen bundle.

## 4.1 Backends

Libyubihsm requires a Connector component to talk to a YubiHSM device. This component can be one of two different types.

### 4.2 HTTP Connector

This kind of Connector is a multiplexer daemon that speaks USB to a YubiHSM device and HTTP to the libyubihsm. This is the component described as the **yubihsm-connector**, see *Connector Reference*.

In order to select this type of backend the connector URL should use the http or https scheme; for example, to use a local HTTP Connector use `http://127.0.0.1:12345`.

### 4.3 USB Connector

This kind of Connector is a direct-access USB backend that talks directly with a YubiHSM device. The USB Connector is built into libyubihsm. This renders it unnecessary to run an additional component (i.e., the external Connector) at the cost of requiring exclusive access to a YubiHSM device.

In order to select this type of backend the connector URL should use the yhusb scheme. For example, to use a local device with serial number 123456 use `yhusb://serial=123456`.
The Python library allows you to interface with a YubiHSM 2 through the Connector service using the Python programming language. It supports both Python 2 and Python 3.

The recommended way to install the library is by using pip inside a virtualenv. To create and activate a virtualenv, just run:

```
$ virtualenv yubihsm
Running virtualenv with interpreter /usr/bin/python3
New python executable in /home/user/yubihsm/bin/python3
Also creating executable in /home/user/yubihsm/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.

$ source yubihsm/bin/activate
(yubihsm) $ pip install yubihsm[http,usb]
Collecting yubihsm-2.0.0
... Successfully installed asn1crypto-0.22.0 cffi-1.10.0 cryptography-1.8.1
enum34-1.1.6 idna-2.5 ipaddress-1.0.18 pycparser-2.17 pyusb-1.0.2
requests-2.13.0 yubihsm-2.0.0
(yubihsm) $
```

Note: The cryptography dependency uses C extensions, and therefore has some build dependencies. For detailed instructions, see: https://cryptography.io/en/latest/installation/

```python
from yubihsm import YubiHsm
from yubihsm.objects import AsymmetricKey
from yubihsm.defs import ALGORITHM, CAPABILITY

# Connect to the Connector and establish a session using the default auth key:
hsm = YubiHsm.connect("http://localhost:12345/connector/api")
session = hsm.create_session_derived(1, "password")

# Create a new EC key for signing:
key = AsymmetricKey.generate(session, 0, "EC Key", 1, CAPABILITY.SIGN_ECDSA, ALGORITHM.EC_P256)

# Sign a message
data = b'Hello world!'
```

(continues on next page)
signature = key.sign_ecdsa(data)

# Delete the key from the YubiHSM 2
key.delete()

# Close session and connection:
session.close()
sm.close()
YubiHSM Wrap is a tool that allows the creation of importable objects offline. This is useful when bootstrapping secrets, for example on an air-gapped computer.

The tool requires an unencrypted Wrap Key in binary format and uses that to wrap objects with given Type, Algorithm, ID, Capabilities and, where applicable, Delegated Capabilities.

For the resulting Object to be successfully imported on a YubiHSM 2, the Wrap Key used by yubiHSM-wrap must already be present on the device.

Currently not all Object Types are supported. Refer to Known Issues and Limitations for more information.
The Key Storage Provider (KSP) for Windows Cryptography API: Next Generation (CNG) has been thoroughly tested with Active Directory Certificate Services (AD CS) plus 2048-bit, 3072-bit, and 4096-bit keys. It also works with other types of keys, but those have not been tested to the same extent.

The following installs the KSP and the Connector Service, using them for AD CS with the default Authentication Key (1) and password (password).

When you run the `Install-AdcsCertificationAuthority` command, you should see the YubiHSM 2 light flash rapidly, because AD CS uses the KSP to generate a 2048-bit key in hardware. For AD CS to work properly, `Restart-Computer` may be needed.

```
PS1> msiexec /i "yubihsm-connector-windows-amd64.msi" /passive ACCEPT=yes
PS1> msiexec /i "yubihsm-cngprovider-windows-amd64.msi" /passive ACCEPT=yes
PS1> Install-WindowsFeature AD-Certificate -Verbose
PS1> Install-AdcsCertificationAuthority -CAType EnterpriseRootCa
    -CryptoProviderName "RSA#YubiHSM Key Storage Provider"
    -KeyLength 2048 -HashAlgorithmName SHA256 -ValidityPeriodYears
    -ValidityPeriodUnits 5
PS1> Install-AdcsOnlineResponder
```

If you are using a different Authentication Key, password, or Connector for the KSP, you can specify them as follows (defaults are shown):

```
PS1> Set-ItemProperty -path HKLM:\SOFTWARE\Yubico\YubiHSM
    -name ConnectorURL -Type String -Value http://127.0.0.1:12345
PS1> Set-ItemProperty -path HKLM:\SOFTWARE\Yubico\YubiHSM
    -name AuthKeysetPassword -Type String -Value password
PS1> Set-ItemProperty -path HKLM:\SOFTWARE\Yubico\YubiHSM
    -name AuthKeysetId -Type DWord -Value 1
```

**Warning:** Design considerations for Key Storage Providers in Windows prevent the direct USB functionality of libyubihsm (Connector URL `yhusb:///`), therefore it is not supported in this version of the YubiHSM KSP.

The default configuration for the connector is: `ProgramData\YubiHSM\yubiHSM-connector.yaml` - Administrator rights are required to access the file.
7.1 Additional Documentation for YubiHSM Key Storage Provider

- For instructions on how to move a software-based key into the YubiHSM 2 for use with the KSP, see Move Software Keys to Key Storage Provider.

- For an example of how to create an HSM-backed code signing certificate for Windows through the KSP, see Creating a Code-Signing Certificate using the Key Storage Provider.

- For more information about status codes, see YubiHSM 2 status codes in Windows.

- For details on how to configure the 32-bit and 64-bit KSP DLLs, please see YubiHSM 2 Windows Deployment Guide.
CREATING A CODE-SIGNING CERTIFICATE USING THE KEY STORAGE PROVIDER

This example will show you how to create a code-signing certificate request using a key generated and stored in the YubiHSM 2 via the Key Storage Provider (KSP). This type of code-signing certificate is appropriate for use with the Microsoft `signtool` utility for digitally signing Windows binaries.

In this example, we will use the command line `certreq` utility. All procedures documented here are available in the Certificate Manager (`certmgr.msc`) MMC snap-in if you prefer to use a GUI.

**Note:** For signing, releases prior to and including the current yubihsm2-sdk release have an 8192-byte file size limit - 4k in interactive mode, or 8k in non-interactive mode.

### 8.1 Configure the Key Storage Provider

By default, the KSP will use the factory authentication key in slot 1. If the factory authentication key no longer exists or a different authentication key is desired, the KSP must first be configured with the desired key ID and password.

**Note:** The configured authentication key must at a minimum have the capabilities `generate-asymmetric-key`, `sign-pkcs` and delegated capability `sign-pkcs`. If you want the generated key to be exportable, then add the `exportable-under-wrap` delegated capability.

#### 8.1.1 Authentication Key Example

Create a new Authentication Key capable of generating exportable asymmetric keys through KSP.

```
yubihsn> put authkey 0 0 "GenerateKey" 1 generate-asymmetric-key,
    sign-pkcs sign-pkcs,exportable-under-wrap password
 Stored Authentication key 0x0e32
```
8.2 Create the Certificate Request Configuration File

To specify your request, the `certreq` utility requires an `.inf` file as input. An example file is supplied here:

```
[Version]
Signature="$Windows NT$

[NewRequest]
Subject = "CN=My Publisher" ; Entity name (dns name/upn for other cert types)
HashAlgorithm = sha256 ; Request uses sha256 hash
KeyAlgorithm = RSA ; Key pair generated using RSA algorithm
Exportable = FALSE ; Private key is not exportable
ExportableEncrypted = FALSE ; Private key is not exportable encrypted
KeyLength = 2048 ; YubiHSM KSP key sizes: 2048, 3072, 4096
KeySpec = 2 ; 1 = AT_KEYEXCHANGE, 2 = AT_SIGNATURE
KeyUsage = 0x80 ; 80 = Digital Signature, 20 = Key Encipherment (bitmask)
MachineKeySet = FALSE ; True: cert belongs the local computer, False: current user
KeyUsageProperty = NCRYPT_ALLOW_SIGNING_FLAG ; Private key only used for signing, not decryption
UseExistingKeySet = FALSE ; Do not use an existing key pair
ProviderName = "YubiHSM Key Storage Provider"
ProviderType = 1
SMIME = FALSE ; No secure email function
UseExistingKeySet = FALSE ; Do not use an existing key pair
RequestType = PKCS10 ; Can be CMC, PKCS10, PKCS7 or Cert (self-signed)

[Strings]
szOID_ENHANCED_KEY_USAGE = "2.5.29.37"
szOID_CODE_SIGN = "1.3.6.1.5.5.7.3.3"
szOID_BASIC_CONSTRAINTS = "2.5.29.19"

[Extensions]
%szOID_ENHANCED_KEY_USAGE% = "{text}%szOID_CODE_SIGN%"
%szOID_BASIC_CONSTRAINTS% = "{text}ca=0&pathlength=0"

; If you are using ADCS with certificate templates, you may add
; a specific template under [RequestAttributes]
;[RequestAttributes]
;CertificateTemplate= CodeSigning
```
8.3 Create the Certificate Request

Once you have created the certificate request configuration file, pass it to `certreq` as the input file argument, e.g:

```
certreq -new sign.inf sign.req
```

8.4 Sign the Certificate Request

In the above example, the certificate request was written to `sign.req`. Take this file and submit its contents to your CA for signature. Once signed, open the resulting file (e.g., `sign.crt`) and install the certificate to your personal store.

8.5 Sign using Signtool

Open a prompt with `signtool` in the path and use the following command to sign your binary.

```
> signtool sign <binary name>
```

If you have multiple certificates available for code signing, it may be necessary to identify your signing certificate by hash. If this occurs, `signtool` will show you a list of valid certificates. Simply re-run sign tool with the `sha1` hash of the certificate:

```
> signtool sign /sha1 <certificate hash> <binary name>
```

When importing the certificate for the first time on a new computer, it may be necessary to manually bind the certificate to the private key. This is because the key is not stored with the certificate and Windows doesn't automatically create an association between the two.

After importing the certificate to your personal store, use the `certutil` utility provided by Windows to associate the YubiHSM private key to the certificate.

```
> certutil -repairstore my <certificate hash>
```

8.6 Troubleshooting

The error messages returned from `signtool` are often unhelpful in diagnosing why a signing operation failed. In these situations there are a few commands you can use to track down the root cause.

When using `signtool`, use the `/v` and `/debug` flags to get more detailed output. The example below shows a response you may receive if the certificate is installed but the YubiHSM is not connected or is misconfigured.

```
> signtool sign /v /debug <binary name>
After EKU filter, 1 certs were left.
After expiry filter, 1 certs were left.
After Hash filter, 1 certs were left.
After Private Key filter, 0 certs were left.
SignTool Error: No certificates were found that met all the given criteria.
```

Use `certutil` to check the validity of the imported certificate.
YubiHSM 2 Component Reference

```bash
> certutil -verifystore my <certificate hash>
================ Certificate 0 ==================
Serial Number: 029fe48291dd587c1e6f42bca341291
... 
Certificate is valid
```

Use `certutil` to check whether the KSP has been installed correctly. You should see Provider Name: YubiHSM Key Storage Provider as one of the entries with no errors.

```bash
> certutil -csplist
... 
Provider Name: YubiHSM Key Storage Provider
... 
```

Use `certutil` to check if the key is accessible through the storage provider. You can also add the `-v` flag to get additional details.

```bash
> certutil -csp "YubiHSM Key Storage Provider" -key
YubiHSM Key Storage Provider:
tq-75c94c4b-5e40-4e44-bcd2-ee3330d4942f
RSA
    AT_SIGNATURE
```

Use `certutil` to dump certificate information. This command may show Cannot find the certificate and private key for decryption. when using a new computer if `certutil -repairstore` hasn’t yet been performed.

```bash
> certutil -store my <certificate hash>
================ Certificate 0 ==================
Serial Number: 029fe48291dd587c1e6f42bca341291
... 
Private key is NOT exportable 
Signature test passed
```

### 8.7 More Information

For a detailed explanation of all options available in the request `.inf` file, see the documentation for the `certreq` utility.

To generate a similar request using the Certificate Manager, open the Certificate Manager snap-in, select the Personal/Certificates store, right click and select All Tasks > Advanced Operations > Create Custom Request.
If the target private key is managed by the Microsoft Software Key Storage Provider, another software provider, or any other KSP that allows export via PKCS#12 PFX, it is possible to move your key to the YubiHSM 2, but results may vary.

This process relies on using the `-repairstore` functionality of the `certutil` command, so the private key must only be present via the YubiHSM Key Storage Provider when performing this step. Please refer to the source storage provider documentation for how to cleanly and completely delete a private key.

Because KSP implementations differ, we recommend testing this procedure using your existing provider before affecting a live system.

### 9.1 Export your Existing Private Key and Certificate

Refer to your current KSP documentation on how to obtain a PKCS#12 PFX export of your certificate and private key.

Once you have obtained your PFX file, split the certificate from the PFX file using `certutil`:

```
PS1> certutil -split -dump <pfx file>
```

This will create a file named `<Cert Hash>.crt`.

If you are moving the key to the YubiHSM 2 on the same machine, you must delete the original private key in your current provider. To do so, first execute

```
PS1> certutil -delkey
```

Locate the key that corresponds with the CA. It may look something like this:

```
Microsoft Software Key Storage Provider:
   EXAMPLE-CA abcdef1234fedcba4321abcdef123456_9cfc1053-1b5a-44d7-8a7e-3a8a1c0d0db0 RSA AT_KEYEXCHANGE
```

To delete this example private key, execute:

```
PS1> certutil -delkey -csp "Microsoft Software Key Storage Provider"
   "abcdef1234fedcba4321abcdef123456_9cfc1053-1b5a-44d7-8a7e-3a8a1c0d0db0"
```
9.2 Import the Target Private Key

Using the instructions for importing a PFX private key via yubihsm-shell, import the target private key file to your YubiHSM 2.

Record the Label property of your imported key.

**Important:** The certutil utility does not provide an easy way to split a key exported from the Software KSP into an unencrypted PEM file. It may be necessary to use another tool like OpenSSL to convert the key file to an unencrypted format for import into the HSM. For example, to export the private key, execute

```
PS1> openssl pkcs12 -in <pfx file> -nocerts -out ca.key -nodes
```

To remove the passphrase from the private key, execute

```
PS1> openssl rsa -in ca.key -out ca.key
```

9.3 Restore the Target Certificate

Move the target certificate file (<Cert Hash>.crt) to the target machine.

Import the certificate to the LocalMachine “My” store via your favorite method. At this point, the certificate will not have an associated private key. We’ll use the -repairstore functionality of certutil to re-associate the certificate to the private key.

Make sure that the target private key is visible via the YubiHSM KSP, using

```
PS1> certutil -key -csp "YubiHSM Key Storage Provider"
```

This command will list all private keys (and their corresponding container names - which are equal to the Label property in the YubiHSM 2) visible to the current Authentication Key.

Open an elevated prompt and execute the command

```
PS1> certutil -repairstore MY <Cert Hash>
```

Verify that the certificate has been associated with the YubiHSM KSP and has the correct Key Container property value by running

```
PS1> certutil -store My
```

and inspecting the Key Container and Provider properties.

**Warning:** If you are moving your CA key to the YubiHSM 2 on the same machine, Windows Certificate Services (CertSvc) on the local machine writes the name of the KSP to its configuration section in the registry. When signing requests, the certificate service will fail if the KSP name does not match the name in the registry.

To update the KSP name for the local certificate service, open an elevated prompt and execute the commands:
If you have multiple CAs on the same machine, or prefer to edit the registry directly, these settings can be found under

```
HKLM\System\CurrentControlSet\Services\CertSVC\Configuration\<CA_Name>\[CSP | EncryptionCSP]
```
STATUS CODES REFERENCE

The YubiHSM software components have a standard set of status codes to report the status of an HSM operation. To comply with the expectations of specific platforms, these status codes are converted to the appropriate API status code. Currently, this translation is only performed for the Windows Key Storage Provider. The error codes, their meanings and translated values are as follows:

<table>
<thead>
<tr>
<th>Libyubihsm Error Code</th>
<th>Description</th>
<th>Windows CNG Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>YHR_BUFFER_TOO_SMALL</td>
<td>Not enough space to store data</td>
<td>NTE_BUFFER_TOO_SMALL</td>
</tr>
<tr>
<td>YHR_CONNECTION_ERROR</td>
<td>Transport Backend error</td>
<td>NTE_DEVICE_NOT_READY</td>
</tr>
<tr>
<td>YHR_CONNECTOR_ERROR</td>
<td>Connector operation failed</td>
<td>NTE_DEVICE_NOT_READY</td>
</tr>
<tr>
<td>YHR_CONNECTOR_NOT_FOUND</td>
<td>Unable to find a suitable connector</td>
<td>NTE_DEVICE_NOT_READY</td>
</tr>
<tr>
<td>YHR_CRYPTOGRAM_MISMATCH</td>
<td>Unable to verify cryptogram</td>
<td>NTE_BAD_SIGNATURE</td>
</tr>
<tr>
<td>YHR_DEVICE_AUTHENTICATION_FAILED</td>
<td>Message encryption / verification failed</td>
<td>NTE_INCORRECT_PASSWORD</td>
</tr>
<tr>
<td>YHR_DEVICE_COMMAND_UNEXECUTED</td>
<td>The HSM attempted to execute a command, but it did not complete in the time allotted. The command has not terminated, and the current state of the session is unavailable</td>
<td>NTE_SYS_ERR</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Libyuhsm Error Code</th>
<th>Description</th>
<th>Windows CNG Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>YHR_DEVICE_DEMO_MODE</td>
<td>Demo mode, power cycle device</td>
<td>NTE_DEVICE_NOT_READY</td>
</tr>
<tr>
<td>YHR_DEVICE_INSUFFICIENT_PERMISSIONS</td>
<td>Wrong permissions for operation</td>
<td>NTE_PERM</td>
</tr>
<tr>
<td>YHR_DEVICE_INVALID_COMMAND</td>
<td>Invalid command</td>
<td>NTE_NOT_SUPPORTED</td>
</tr>
<tr>
<td>YHR_DEVICE_INVALID_DATA</td>
<td>Malformed command / invalid data</td>
<td>NTE_INVALID_PARAMETER</td>
</tr>
<tr>
<td>YHR_DEVICE_INVALID_ID</td>
<td>Illegal ID used</td>
<td>NTE_INVALID_ID</td>
</tr>
<tr>
<td>YHR_DEVICE_INVALID_OTP</td>
<td>Invalid OTP</td>
<td>NTE_INCORRECT_PASSWORD</td>
</tr>
<tr>
<td>YHR_DEVICE_INVALID_SESSION</td>
<td>Invalid session</td>
<td>NTE_DEVICE_NOT_READY</td>
</tr>
<tr>
<td>YHR_DEVICE_LOG_FULL</td>
<td>Log buffer is full and forced audit is set</td>
<td>NTE_DEVICE_NOT_READY</td>
</tr>
<tr>
<td>YHR_DEVICE_OBJECT_EXISTS</td>
<td>An object with the specified ID already exists</td>
<td>NTE_EXISTS</td>
</tr>
<tr>
<td>YHR_DEVICE_OBJECT_NOT_FOUND</td>
<td>Object not found</td>
<td>NTE_NOT_FOUND</td>
</tr>
<tr>
<td>YHR_DEVICE_OK</td>
<td>No error</td>
<td>NTE_OP_OK</td>
</tr>
<tr>
<td>YHR_DEVICE_SESSION_FAILED</td>
<td>Session creation failed</td>
<td>NTE_DEVICE_NOT_READY</td>
</tr>
<tr>
<td>YHR_DEVICE_SESSIONS_FULL</td>
<td>All sessions are allocated</td>
<td>NTE_DEVICE_NOT_READY</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Libyubihs Error Code</th>
<th>Description</th>
<th>Windows CNG Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>YHR_DEVICE_STORAGE_FAILED</td>
<td>Storage failure</td>
<td>NTE_TOKEN_KEYSET _STORAGE_FULL</td>
</tr>
<tr>
<td>YHR_DEVICE_WRONG_LENGTH</td>
<td>Wrong length</td>
<td>NTE_BAD_LEN</td>
</tr>
<tr>
<td>YHR_GENERIC_ERROR</td>
<td>Generic error</td>
<td>NTE_FAIL</td>
</tr>
<tr>
<td>YHR_INIT_ERROR</td>
<td>Unable to initialize libyubihs</td>
<td>NTE_PROVIDER_DLL_FAIL</td>
</tr>
<tr>
<td>YHR_INVALID_PARAMETERS</td>
<td>Invalid argument to a function</td>
<td>NTE_INVALID_PARAMETER</td>
</tr>
<tr>
<td>YHR_MAC_MISMATCH</td>
<td>Unable to verify MAC</td>
<td>NTE_BAD_SIGNATURE</td>
</tr>
<tr>
<td>YHR_MEMORY_ERROR</td>
<td>The YubiHSM or software library was not able to allocate memory to</td>
<td>NTE_NO_MEMORY</td>
</tr>
<tr>
<td></td>
<td>perform the requested operation</td>
<td></td>
</tr>
<tr>
<td>YHR_SESSION _AUTHENTICATION_FAILED</td>
<td>Unable to authenticate session</td>
<td>NTE_INCORRECT_PASSWORD</td>
</tr>
<tr>
<td>YHR_SUCCESS</td>
<td>The operation completed successfully</td>
<td>ERROR_SUCCESS</td>
</tr>
<tr>
<td>YHR_WRONG_LENGTH</td>
<td>This error may occur if there is a mismatch between the YubiHSM firmware</td>
<td>NTE_BAD_LEN</td>
</tr>
<tr>
<td></td>
<td>version and libyubihs library version</td>
<td></td>
</tr>
</tbody>
</table>
11.1 Configuration

The PKCS#11 module requires a configuration file, default location for this file is current directory and default name is *yubihsm_pkcs11.conf* using the environment variable *YUBIHSM_PKCS11_CONF* one can point to a custom location and name.

Configuration options can also be passed as a string in the *pReserved* field of *C.Initialize*, using the OpenSSL PKCS#11 engine this can be set in the *INIT_ARGS* configuration value. This is technically a violation of the PKCS#11 specification (which mandates *pReserved* to be set to *NULL*) and is not supported by all applications.

Accepted configuration options:

- **connector**: URL pointing at the connector to contact, mandatory
- **debug**: Turn on PKCS#11 debugging, default off
- **dinout**: Turn on call tracing, default off
- **ibdebug**: Turn on debug of libyubihsm, default off
- **debug-file**: File to write debug information to, default stderr
- **cacert**: File with cacert to verify connector https cert with (not available on Windows)
- **proxy**: Proxy server for reaching the connector (not available on Windows)
- **timeout**: Timeout to use for initial connection to the connector (in seconds), default 5

A *Configuration File Sample* can be found below.

11.2 Logging In

All interesting operations through the PKCS#11 interface require a logged-in session, and one peculiarity of the PKCS#11 interface is that the user PIN **MUST** be prefixed by the ID (16 bits, in hexadecimal, zero padded if required) of the corresponding Authentication Key.

Assuming the default Authentication Key with ID 1 and password password, the user PIN would then be 0001password. To be compliant with PKCS#11 standards, the Authentication Key password **MUST** be at least 8 characters long.

Note that the concept of a Security Officer (SO) is not supported by the device, and the PIN management functions are not implemented for neither user nor SO.

It is recommended that PIN (Authentication Key) management is performed via the *yubihsms-shell* utility or the *libyubihsm* functions.
11.3 PKCS#11 on Windows

After installing yubihsms-shell using the windows installer, in addition to setting YUBIHSM_PKCS11_CONF environment variable, the YubiHSM Shell\bin directory needs to be added to the system path in order for other applications to be able to load it. This is because the yubihsms-pkcs11.dll is dynamically linked to the libyubihsms*.dll and libcrypto-1_1.dll libraries and they need to be accessible for the PKCS#11 module to be useful.

On Windows 10, setting the system path is done by following these steps:

**Step 1** Go to Control Panel > System and Security > System > Advanced system setting
**Step 2** Click Environment Variables…
**Step 3** Under System Variables, highlight Path and click Edit…
**Step 4** Click New and add the absolute path to YubiHSM Shell/bin
**Step 5** Under System Variables, click New and add the environment variable YUBIHSM_PKCS11_CONF and set it to the path to the YubiHSM2 PKCS11 configuration file

If setting the system path is not desirable, the libyubihsms*.dll and libcrypto-1_1.dll can be copied into the same directory as the application that needs to access the PKCS#11 module.

11.3.1 Note for Developers

If LoadLibrary is called with an absolute path, it will not look for dependencies of the specified DLL in that directory, but rather in the startup directory of the application that calls LoadLibrary. The solution is to either:

- Call LoadLibraryEx with the flag LOAD_WITH_ALTERED_SEARCH_PATH for absolute paths
- Add the directory where the PKCS#11 module is located to the system PATH
- Or copy the dependencies into the application directory.

Please note that calling LoadLibraryEx with that flag for a non-absolute path is undefined behavior according to MS docs. For example, the way Pkcs11Interop does it is to set a variable to LOAD_WITH_ALTERED_SEARCH_PATH if the path looks absolute, and 0 otherwise; and then always calling LoadLibraryEx. If the flags is 0 then LoadLibraryEx behaves exactly like LoadLibrary.

11.3.2 Software Operations

C_Encrypt and C_Verify for Asymmetric Keys are performed in software, as well as all of the C_Digest operations.

11.4 PKCS#11 Attributes

There are a number of attributes defined in PKCS#11 that do not translate to Capabilities of the YubiHSM 2 device, and are therefore treated as always having a fixed value.
11.4.1 Capabilities and Domains

Objects created via the PKCS#11 module inherit the Domains of the Authentication Key used to establish the session. The Domains cannot be changed or modified via the module.

Object Capabilities are set on creation, depending on their Type, e.g. an RSA signing key (CKK_RSA) created via C_CreateObject with the attribute CKA_SIGN set will have the following Capabilities set sign-pkcs, sign-pss.

Similarly for EC (CKK_EC), the key would have sign-ecdsa set.

See the following tables for mappings:

<table>
<thead>
<tr>
<th>PKCS#11</th>
<th>RSA (CKK_RSA)</th>
<th>EC (CKK_EC)</th>
<th>Wrap (CKK_YUBICO_AES* _CCM_WRAP)</th>
<th>HMAC (CKK_SHA* _HMAC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_ENCRYPT</td>
<td>N/A</td>
<td>N/A</td>
<td>wrap-data</td>
<td>N/A</td>
</tr>
<tr>
<td>CKA_EXTRACTABLE</td>
<td>export-under-wrap</td>
<td>export-under-wrap</td>
<td>export-under-wrap</td>
<td>export-under-wrap</td>
</tr>
<tr>
<td>CKA_DECRYPT</td>
<td>decrypt-pkcs, decrypt-oaep</td>
<td>N/A</td>
<td>unwrap-data</td>
<td>N/A</td>
</tr>
<tr>
<td>CKA_DERIVE</td>
<td>N/A</td>
<td>derive-ecdh</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>CKA_SIGN</td>
<td>sign-pkcs, sign-pss</td>
<td>sign-ecdsa</td>
<td>N/A</td>
<td>sign-hmac</td>
</tr>
<tr>
<td>CKA_VERIFY</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>verify-hmac</td>
</tr>
<tr>
<td>CKA_WRAP</td>
<td>N/A</td>
<td>N/A</td>
<td>export-wrapped</td>
<td>N/A</td>
</tr>
<tr>
<td>CKA_UNWRAP</td>
<td>N/A</td>
<td>N/A</td>
<td>import-wrapped</td>
<td>N/A</td>
</tr>
</tbody>
</table>

11.5 PKCS#11 Objects

Not all PKCS#11 Object types are implemented, this is a list of what is implemented and what it maps to.
### 11.6 PKCS#11 Functions

Not all functions in PKCS#11 are implemented in the module, this is a list of what is implemented.

<table>
<thead>
<tr>
<th>PKCS#11</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_CloseSession</td>
<td></td>
</tr>
<tr>
<td>C_CloseAllSessions</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>PKCS#11</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_CreateObject</td>
<td>Use with CKO_PRIVATE_KEY, CKO_SECRET_KEY, CKO_CERTIFICATE or CKO_DATA</td>
</tr>
<tr>
<td>C_Decrypt</td>
<td></td>
</tr>
<tr>
<td>C_DecryptFinal</td>
<td></td>
</tr>
<tr>
<td>C_DecryptInit</td>
<td>Decrypt with Wrap Key or RSA key</td>
</tr>
<tr>
<td>C_DecryptUpdate</td>
<td></td>
</tr>
<tr>
<td>C_DeriveKey</td>
<td>Derive key using ECDH as a PKCS#11 session object</td>
</tr>
<tr>
<td>C_DestroyObject</td>
<td></td>
</tr>
<tr>
<td>C_Digest</td>
<td></td>
</tr>
<tr>
<td>C_DigestFinal</td>
<td></td>
</tr>
<tr>
<td>C_DigestInit</td>
<td>Do software digest with CKM_SHA_1, CKM_SHA256, CKM_SHA384 or CKM_SHA512</td>
</tr>
<tr>
<td>C_DigestUpdate</td>
<td></td>
</tr>
<tr>
<td>C_Encrypt</td>
<td></td>
</tr>
<tr>
<td>C_EncryptFinal</td>
<td></td>
</tr>
<tr>
<td>C_EncryptInit</td>
<td>Encrypt with Wrap Key or do software encryption for RSA key</td>
</tr>
<tr>
<td>C_EncryptUpdate</td>
<td></td>
</tr>
<tr>
<td>C_Finalize</td>
<td></td>
</tr>
<tr>
<td>C_FindObjects</td>
<td></td>
</tr>
<tr>
<td>C_FindObjectsFinal</td>
<td></td>
</tr>
<tr>
<td>C_FindObjectsInit</td>
<td></td>
</tr>
<tr>
<td>C_GenerateKey</td>
<td>Generate HMAC Key or Wrap Key</td>
</tr>
<tr>
<td>C_GenerateKeyPair</td>
<td>Generate Asymmetric Key</td>
</tr>
<tr>
<td>C_GenerateRandom</td>
<td>Generate up to 2021 bytes of random</td>
</tr>
<tr>
<td>C_GetAttributeValue</td>
<td></td>
</tr>
<tr>
<td>C_GetFunctionList</td>
<td></td>
</tr>
<tr>
<td>C_GetInfo</td>
<td></td>
</tr>
<tr>
<td>C_GetMechanismList</td>
<td></td>
</tr>
<tr>
<td>C_GetMechanismInfo</td>
<td></td>
</tr>
<tr>
<td>C.GetObjectSize</td>
<td></td>
</tr>
<tr>
<td>C_GetSessionInfo</td>
<td></td>
</tr>
<tr>
<td>C_GetSlotInfo</td>
<td></td>
</tr>
<tr>
<td>C_GetSlotList</td>
<td></td>
</tr>
<tr>
<td>C_GetTokenInfo</td>
<td></td>
</tr>
<tr>
<td>C_Initialize</td>
<td></td>
</tr>
<tr>
<td>C_Login</td>
<td></td>
</tr>
<tr>
<td>C_Logout</td>
<td></td>
</tr>
<tr>
<td>C_OpenSession</td>
<td></td>
</tr>
<tr>
<td>C_Sign</td>
<td></td>
</tr>
<tr>
<td>C_SignFinal</td>
<td>continues on next page</td>
</tr>
</tbody>
</table>

11.6. PKCS#11 Functions
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>PKCS#11</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_SignInit</td>
<td>Sign with HMAC Key or Asymmetric Key</td>
</tr>
<tr>
<td>C_SignUpdate</td>
<td></td>
</tr>
<tr>
<td>C_Verify</td>
<td></td>
</tr>
<tr>
<td>C_VerifyFinal</td>
<td></td>
</tr>
<tr>
<td>C_VerifyInit</td>
<td>Verify HMAC or software verify asymmetric</td>
</tr>
<tr>
<td>C_VerifyUpdate</td>
<td></td>
</tr>
<tr>
<td>C_UnwrapKey</td>
<td>Unwrap an object with Wrap Key</td>
</tr>
<tr>
<td>C_WrapKey</td>
<td>Wrap an object with Wrap Key</td>
</tr>
</tbody>
</table>

11.7 PKCS#11 Vendor Definitions

Working with the device Wrap Keys requires using vendor-specific definitions, these are listed in the table below. The Wrap Keys can be used with C_WrapKey, C_UnwrapKey, C_Encrypt & C_Decrypt.

<table>
<thead>
<tr>
<th>Wrap Type</th>
<th>Wrap Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKM_YUBICO_AES_CCM_WRAP</td>
<td>0xd9554204</td>
</tr>
<tr>
<td>CKK_YUBICO_AES128_CCM_WRAP</td>
<td>0xd955421d</td>
</tr>
<tr>
<td>CKK_YUBICO_AES192_CCM_WRAP</td>
<td>0xd9554229</td>
</tr>
<tr>
<td>CKK_YUBICO_AES256_CCM_WRAP</td>
<td>0xd955422a</td>
</tr>
</tbody>
</table>

11.8 PKCS#11 Configuration

11.8.1 Configuration File Sample

Below is a sample of a yubihsm_pkcs11.conf configuration file.

```bash
# This is a sample configuration file for the YubiHSM PKCS#11 module
# Uncomment the various options as needed

# URL of the connector to use. This can be a comma-separated list
connector = http://127.0.0.1:12345

# Enables general debug output in the module
# debug

# Enables function tracing (ingress/egress) debug output in the module
# dinout

# Enables libyubihsm debug output in the module
# libdebug

# Redirects the debug output to a specific file. The file is created
# if it does not exist. The content is appended
```

(continues on next page)
11.8.2 INIT_ARGS Sample

Below is a sample of using the INIT_ARGS configuration with an openssl.cnf file.

```
openssl_conf = openssl_init

[openssl_init]
engines = engine_section

[engine_section]
pkcs11 = pkcs11_section

pkcs11_section]
engine_id = pkcs11
dynamic_path = /path/to/engine_pkcs11.so
MODULE_PATH = /path/to/yubihsmpkcs11.so
INIT_ARGS = connector=http://127.0.0.1:12345 debug
init = 0
```

Note: OpenSSL 1.1 will auto-load modules present in the system engine directory (like /usr/lib/x86_64-linux-gnu/engines-1.1) so the dynamic_path line has to be dropped there. The error shown will mention “conflicting engine id”.

---

```
# debug-file = /tmp/yubihsm_pkcs11_debug

# CA certificate to use for HTTPS validation. Point this variable to
# a file containing one or more certificates to use when verifying
# a peer. Currently not supported on Windows
#
# cacert = /tmp/cacert.pem

# Proxy server to use for the connector
# Currently not supported on Windows
#
# proxy = http://proxyserver.local.com:8080

# Timeout in seconds to use for the initial connection to the connector
# timeout = 5
```
CHAPTER

TWELVE

PKCS#11 TOOL COMPATIBILITY, INTEROPERABILITY AND KNOWN RESTRICTIONS

This topic contains information about the different tools that are either known to work or known not to work with the current version of the YubiHSM 2.

12.1 pkcs11-tool

This is the tool produced by OpenSC.

Running with HEAD on master (currently dfd18389346296f8e4617832e0d5f4171835620d) the command used is

```
pkcs11-tool --module yubihsm_pkcs11.so -l -p 0001password -t
```

All relevant tests are passing with the following notable exceptions:
- RSA-PKCS-OAEP decryption: the test appears to be broken. It calls into OpenSSL's EVP_PKEY_encrypt/EVP_PKEY_encrypt_old which uses PKCS1v1.5 padding.
- mechtype-0xD9554204 decryption: this a Yubico custom mechanism (AES-CCM wrapping) and can't be handled by the tool.

12.2 pkcs11test

This is a PKCS#11 tester tool by Google. It is built as a test target in the source code. We maintain an internal version to accommodate some differences at https://github.com/Yubico/pkcs11test.

The command used

```
pkcs11test -myubihsm_pkcs11.so -l -u0001password --gtest_filter=-${SKIPPED_TESTS_STR}
```

where SKIPPED_TESTS_STR is the list below.

All relevant tests pass. The following tests have been explicitly skipped:

```
Slot.NoInit
PKCS11Test.EnumerateMechanisms
ReadOnlySessionTest.GenerateRandom
ReadOnlySessionTest.GenerateRandomNone
ReadOnlySessionTest.UserLoginWrongPIN
ReadOnlySessionTest.SOLoginFail
ReadOnlySessionTest.CreateKeyPairObjects
```

(continues on next page)
YubiHSM 2 Component Reference

ReadOnlySessionTest.CreateSecretKeyAttributes
ReadOnlySessionTest.SecretKeyTestVectors
ReadOnlySessionTest.SignVerifyRecover
ReadOnlySessionTest.GenerateKeyInvalid
ReadOnlySessionTest.GenerateKeyPairInvalid
ReadOnlySessionTest.WrapUnwrap
ReadOnlySessionTest.WrapInvalid
ReadOnlySessionTest.UnwrapInvalid
ReadWriteSessionTest.CreateCopyDestroyObject
ReadWriteSessionTest.SetLatchingAttribute
ReadWriteSessionTest.FindObjectSubset
ReadWriteSessionTest.ReadOnlySessionSOLoginFail
ReadWriteSessionTest.SOLogin
ReadWriteSessionTest.TookanAttackA1
ReadWriteSessionTest.TookanAttackA3
ReadWriteSessionTest.TookanAttackA4
ReadWriteSessionTest.TookanAttackA5a
ReadWriteSessionTest.TookanAttackA5b
ReadWriteSessionTest.PublicExponent4Bytes
ReadWriteSessionTest.ExtractKeys
ReadWriteSessionTest.AsymmetricTokenKeyPair
RWUserSessionTest.SOLoginFail
DataObjectTest.CopyDestroyObjectInvalid
DataObjectTest.GetMultipleAttributes
DataObjectTest.GetUserLoginFail
RWSOSessionTest.SOSessionFail
RWSOSessionTest.UserLoginFail
RWEitherSessionTest.TookanAttackA2
KeyPairTest.EncryptDecrypt
Ciphers/SecretKeyTest.EncryptDecrypt/0
Ciphers/SecretKeyTest.EncryptDecrypt/1
Ciphers/SecretKeyTest.EncryptDecrypt/2
Ciphers/SecretKeyTest.EncryptDecrypt/3
Ciphers/SecretKeyTest.EncryptDecrypt/4
Ciphers/SecretKeyTest.EncryptDecrypt/5
Ciphers/SecretKeyTest.EncryptFailDecrypt/0
Ciphers/SecretKeyTest.EncryptFailDecrypt/1
Ciphers/SecretKeyTest.EncryptFailDecrypt/2
Ciphers/SecretKeyTest.EncryptFailDecrypt/3
Ciphers/SecretKeyTest.EncryptFailDecrypt/4
Ciphers/SecretKeyTest.EncryptFailDecrypt/5
Ciphers/SecretKeyTest.EncryptDecryptGetSpace/0
Ciphers/SecretKeyTest.EncryptDecryptGetSpace/1
Ciphers/SecretKeyTest.EncryptDecryptGetSpace/2
Ciphers/SecretKeyTest.EncryptDecryptGetSpace/3
Ciphers/SecretKeyTest.EncryptDecryptGetSpace/4
Ciphers/SecretKeyTest.EncryptDecryptGetSpace/5
Ciphers/SecretKeyTest.EncryptDecryptParts/0
Ciphers/SecretKeyTest.EncryptDecryptParts/1
Ciphers/SecretKeyTest.EncryptDecryptParts/2
Ciphers/SecretKeyTest.EncryptDecryptParts/3
Ciphers/SecretKeyTest.EncryptDecryptParts/4

(continues on next page)
| Ciphers/SecretKeyTest.EncryptDecryptParts/5 |
| Ciphers/SecretKeyTest.EncryptDecryptInitInvalid/0 |
| Ciphers/SecretKeyTest.EncryptDecryptInitInvalid/1 |
| Ciphers/SecretKeyTest.EncryptDecryptInitInvalid/2 |
| Ciphers/SecretKeyTest.EncryptDecryptInitInvalid/3 |
| Ciphers/SecretKeyTest.EncryptDecryptInitInvalid/4 |
| Ciphers/SecretKeyTest.EncryptDecryptInitInvalid/5 |
| Ciphers/SecretKeyTest.EncryptErrors/0 |
| Ciphers/SecretKeyTest.EncryptErrors/1 |
| Ciphers/SecretKeyTest.EncryptErrors/2 |
| Ciphers/SecretKeyTest.EncryptErrors/3 |
| Ciphers/SecretKeyTest.EncryptErrors/4 |
| Ciphers/SecretKeyTest.EncryptErrors/5 |
| Ciphers/SecretKeyTest.DecryptErrors/0 |
| Ciphers/SecretKeyTest.DecryptErrors/1 |
| Ciphers/SecretKeyTest.DecryptErrors/2 |
| Ciphers/SecretKeyTest.DecryptErrors/3 |
| Ciphers/SecretKeyTest.DecryptErrors/4 |
| Ciphers/SecretKeyTest.DecryptErrors/5 |
| Ciphers/SecretKeyTest.EncryptUpdateErrors/0 |
| Ciphers/SecretKeyTest.EncryptUpdateErrors/1 |
| Ciphers/SecretKeyTest.EncryptUpdateErrors/2 |
| Ciphers/SecretKeyTest.EncryptUpdateErrors/3 |
| Ciphers/SecretKeyTest.EncryptUpdateErrors/4 |
| Ciphers/SecretKeyTest.EncryptUpdateErrors/5 |
| Ciphers/SecretKeyTest.EncryptModePolicing1/0 |
| Ciphers/SecretKeyTest.EncryptModePolicing1/1 |
| Ciphers/SecretKeyTest.EncryptModePolicing1/2 |
| Ciphers/SecretKeyTest.EncryptModePolicing1/3 |
| Ciphers/SecretKeyTest.EncryptModePolicing1/4 |
| Ciphers/SecretKeyTest.EncryptModePolicing1/5 |
| Ciphers/SecretKeyTest.EncryptModePolicing2/0 |
| Ciphers/SecretKeyTest.EncryptModePolicing2/1 |
| Ciphers/SecretKeyTest.EncryptModePolicing2/2 |
| Ciphers/SecretKeyTest.EncryptModePolicing2/3 |
| Ciphers/SecretKeyTest.EncryptModePolicing2/4 |
| Ciphers/SecretKeyTest.EncryptModePolicing2/5 |
| Ciphers/SecretKeyTest.EncryptInvalidIV/0 |
| Ciphers/SecretKeyTest.EncryptInvalidIV/1 |
| Ciphers/SecretKeyTest.EncryptInvalidIV/2 |
| Ciphers/SecretKeyTest.EncryptInvalidIV/3 |
| Ciphers/SecretKeyTest.EncryptInvalidIV/4 |
| Ciphers/SecretKeyTest.EncryptInvalidIV/5 |
| Ciphers/SecretKeyTest.DecryptInvalidIV/0 |
| Ciphers/SecretKeyTest.DecryptInvalidIV/1 |
| Ciphers/SecretKeyTest.DecryptInvalidIV/2 |
| Ciphers/SecretKeyTest.DecryptInvalidIV/3 |
| Ciphers/SecretKeyTest.DecryptInvalidIV/4 |
| Ciphers/SecretKeyTest.DecryptInvalidIV/5 |
| Ciphers/SecretKeyTest.DecryptUpdateErrors/0 |

(continues on next page)
Ciphers/SecretKeyTest.DecryptUpdateErrors/1
Ciphers/SecretKeyTest.DecryptUpdateErrors/2
Ciphers/SecretKeyTest.DecryptUpdateErrors/3
Ciphers/SecretKeyTest.DecryptUpdateErrors/4
Ciphers/SecretKeyTest.DecryptUpdateErrors/5
Ciphers/SecretKeyTest.EncryptFinalImmediate/0
Ciphers/SecretKeyTest.EncryptFinalImmediate/1
Ciphers/SecretKeyTest.EncryptFinalImmediate/2
Ciphers/SecretKeyTest.EncryptFinalImmediate/3
Ciphers/SecretKeyTest.EncryptFinalImmediate/4
Ciphers/SecretKeyTest.EncryptFinalImmediate/5
Ciphers/SecretKeyTest.EncryptFinalErrors1/0
Ciphers/SecretKeyTest.EncryptFinalErrors1/1
Ciphers/SecretKeyTest.EncryptFinalErrors1/2
Ciphers/SecretKeyTest.EncryptFinalErrors1/3
Ciphers/SecretKeyTest.EncryptFinalErrors1/4
Ciphers/SecretKeyTest.EncryptFinalErrors1/5
Ciphers/SecretKeyTest.EncryptFinalErrors2/0
Ciphers/SecretKeyTest.EncryptFinalErrors2/1
Ciphers/SecretKeyTest.EncryptFinalErrors2/2
Ciphers/SecretKeyTest.EncryptFinalErrors2/3
Ciphers/SecretKeyTest.EncryptFinalErrors2/4
Ciphers/SecretKeyTest.EncryptFinalErrors2/5
Ciphers/SecretKeyTest.DecryptFinalErrors1/0
Ciphers/SecretKeyTest.DecryptFinalErrors1/1
Ciphers/SecretKeyTest.DecryptFinalErrors1/2
Ciphers/SecretKeyTest.DecryptFinalErrors1/3
Ciphers/SecretKeyTest.DecryptFinalErrors1/4
Ciphers/SecretKeyTest.DecryptFinalErrors1/5
Ciphers/SecretKeyTest.DecryptFinalErrors2/0
Ciphers/SecretKeyTest.DecryptFinalErrors2/1
Ciphers/SecretKeyTest.DecryptFinalErrors2/2
Ciphers/SecretKeyTest.DecryptFinalErrors2/3
Ciphers/SecretKeyTest.DecryptFinalErrors2/4
Ciphers/SecretKeyTest.DecryptFinalErrors2/5
Digests/DigestTest.DigestKey/0
Digests/DigestTest.DigestKey/1
Digests/DigestTest.DigestKey/2
Digests/DigestTest.DigestKey/3
Digests/DigestTest.DigestKey/4
Digests/DigestTest.DigestKeyInvalid/0
Digests/DigestTest.DigestKeyInvalid/1
Digests/DigestTest.DigestKeyInvalid/2
Digests/DigestTest.DigestKeyInvalid/3
Digests/DigestTest.DigestKeyInvalid/4
Signatures/SignTest.SignVerify/0
Signatures/SignTest.SignFailVerifyWrong/0
Signatures/SignTest.SignFailVerifyShort/0
Duals/DualSecretKeyTest.DigestEncrypt/0
Duals/DualSecretKeyTest.DigestEncrypt/1
Duals/DualSecretKeyTest.DigestEncrypt/2
Duals/DualSecretKeyTest.DigestEncrypt/3
12.2.1 python-pkcs11tester

This is a Yubico tool, developed to run additional tests.
The command used is

```
python setup.py test
```

All relevant tests pass.

12.2.2 p11tool

This is a tool shipped with GnuTLS. From version 3.5.2 it can work with the YubiHSM 2. Keys can be generated like

```
p11tool --provider=yubihsm_pkcs11.so "pkcs11:pin-value=0001password"
    --login --generate-rsa --label="rsa test key" --bits=2048
```

and signatures tested and verified with the command

```
p11tool --provider=yubihsm_pkcs11.so "pkcs11:pin-value=0001password;object=rsakey" --login --test-sign
```

12.2.3 OpenDNSSEC

OpenDNSSEC contains a libhsm and two tools, ods-hsmutil and ods-hsmspeed, both of these work with the YubiHSM 2 with a small configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <RepositoryList>
        <Repository name="default">
            <Module>yubihsm_pkcs11.so</Module>
            <TokenLabel>YubiHSM</TokenLabel>
            <PIN>0001password</PIN>
        </Repository>
    </RepositoryList>
</Configuration>
```

Using this, it is possible to run through tests with the command

```
ods-hsmutil -c conf-yubihsm.xml test default
```

This passes all tests using algorithms supported by the YubiHSM 2 (rsa2048, rsa4096, ecp256, ecp384 & randomness).
CHAPTER
THIRTEEN

COPYRIGHT

© 2022 Yubico AB. All rights reserved.

Trademarks
Yubico and YubiKey are registered trademarks of Yubico AB. All other trademarks are the property of their respective owners.

Disclaimer
The contents of this document are subject to revision without notice due to continued progress in methodology, design, and manufacturing. Yubico shall have no liability for any error or damages of any kind resulting from the use of this document.

The Yubico Software referenced in this document is licensed to you under the terms and conditions accompanying the software or as otherwise agreed between you or the company that you are representing.

Contact Information

Yubico Inc.
530 Lytton Street
Suite 301
Palo Alto, CA 94301
USA

Click the links to:
• Submit a support request
• Send a Contact Me request
• See additional contact options for getting touch with us

Document Updated
2022-05-13 01:39:38 UTC