Azure Cloud HSM SSL/TLS Offloading Integration Guide

Table of Contents

Summary

To use OpenSSL with Azure Cloud HSM for SSL/TLS Offloading for Apache or NGINX, the Azure Cloud HSM Client SDK must be installed, and OpenSSL must be installed and configured on the same host that will be performing SSL/TLS Offloading operations.

Important Note: Azure Cloud HSM OpenSSL Engine does not support Windows. Azure Cloud HSM OpenSSL Engine supports Linux only. SSL/TLS offloading for Azure Cloud HSM is supported through its OpenSSL engine only. The requirement of OpenSSL supporting PKCS#11 for SSL/TLS offloading is not supported by Azure Cloud HSM. Customers that require PKCS#11 for SSL/TLS offloading must use the TLS Offload Library for Azure Managed HSM.

Prerequisites

The following prerequisites are required to support the Azure Cloud HSM OpenSSL engine. Please reference the Azure Cloud HSM Onboarding Guide for SDK Installation and Azure Cloud HSM configuration if you have not completed your HSM deployment.

System Requirements

- Supported Operating Systems: Ubuntu 20.04, Ubuntu 22.04, RHEL 7, RHEL 8, CBL Mariner 2
- Azure Cloud HSM resource has been deployed, initialized, and configured.
- Azure Cloud HSM Client SDK
- Copy of partition owner certificate "PO.crt" on host/signing server.
- Known address of your HSM "hsm1.chsm-<resourcename>-<uniquestring>.privatelink.cloudhsm.azure.net".
- Creation of User-Generated KEK
- Knowledge of Crypto User credentials

Customers can download the Azure Cloud HSM SDK and Client Tools from GitHub:
@ microsoft/MicrosoftAzureCloudHSM: Azure Cloud HSM SDK

1. **Install OpenSSL for Linux**: To install OpenSSL on a Linux system, you'll need to use your system's package manager. The exact command may vary depending on your Linux distribution.

    - **For Red Hat based (using yum):**

```
sudo yum install openssl
```

    - **For Red Hat based systems (using dnf):**

```
sudo dnf install openssl
```

    - **For Ubuntu based systems (using apt):**

```
sudo apt update
sudo apt install openssl
```

2. **Validate OpenSSL is Installed.** You can check if OpenSSL is installed on a Linux system by opening a terminal and entering the following command. If OpenSSL is installed, this command will display the version number of OpenSSL that is currently installed on your system. If OpenSSL is not installed, the command will likely result in an error message.

```
openssl version
```

3. **Set Environment Variables:** Add the following system environment variables to your Linux Server. You may also choose to set permanent environment variables in Linux. You can typically modify configuration files specific to your shell (bash, zsh, etc) or make system-wide changes in profile scripts. You will need to update your environment variables to reflect the correct SDK version running.

   Recommended (Permanent Environment Variables): For production, make environment variables persistent across terminal sessions by adding them to your shell's configuration file. This example uses Bash.

   a. From the terminal edit the .bashrc file (for non-login shells) or .bash_profile (for login shells). You can use any text editor.

   ```
   vim ~/.bashrc
   ```

   or

   ```
   vim ~/.bash_profile
   ```

   b. Add the following lines to the end of the file

   ```
   export azcloudhsm_password="cu1:user1234"
   export azcloudhsm_partition="PARTITION_1"
   export azcloudhsm_openssl_conf=/opt/azurecloudhsm/bin/azcloudhsm_openssl_dynamic.conf
   export LD_LIBRARY_PATH=/opt/azurecloudhsm/lib64/:$LD_LIBRARY_PATH
   ```

   c. Save the file and exit the editor

   d. To apply the changes immediately, run the following.

   ```
   source ~/.bashrc
   ```

   or

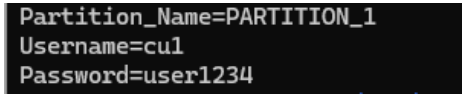
   ```
   source ~/.bash_profile
   ```

   Manual (Testing Purposes Only):
   The following manual approach is temporary and will not persist after the terminal session ends. It should only be used for testing purposes.

```
export azcloudhsm_password="cu1:user1234"
export azcloudhsm_partition="PARTITION_1"
export azcloudhsm_openssl_conf=/opt/azurecloudhsm/bin/azcloudhsm_openssl_dynamic.conf
export LD_LIBRARY_PATH=/opt/azurecloudhsm/lib64/:$LD_LIBRARY_PATH
```

4. **Update sample dynamic configuration file for OpenSSL.** Azure Cloud HSM offers a sample dynamic configuration file for OpenSSL. Please modify the 'Partition_Name' to 'PARTITION_1' and update the 'Username' and 'Password' accordingly. This configuration file will be referenced in the next step when we set environment variable for azcloudhsm_openssl_conf.

```
cd /opt/azurecloudhsm/bin
sudo vim azcloudhsm_openssl_dynamic.conf
```



Creating a User Generated KEK

Creating a User-Generated KEK is necessary only for customers who require Key Import support or integration with JCE and OpenSSL. If you don't need support for those specific use cases, you can choose to skip the creation of a KEK.

1. Creating a user KEK handle involves initial steps where customers execute the azcloudhsm_client, connect to their Cloud HSM using the azcloudhsm_util, and subsequently log in to their HSM. The azcloudhsm_client must be running for azcloudhsm_util to execute. The provided example illustrates the process of generating a user KEK and obtaining its handle.

Important Note: To ensure the proper functioning of the Azure Cloud HSM SDK, customers need to create a user KEK. The functionality of OpenSSL, JCE, and other features within Azure Cloud HSM relies on the existence of a user-generated KEK. Failure to generate a user KEK will result in operational issues.

Important Note: Please keep track of your Key Handle ID generated. You'll require this Key Handle ID to finalize the process. If you need to perform Key Import (key wrap/unwrap), ensure your user KEK is extractable and designated as trusted. Failure to designate your user KEK as extractable and trusted prior to attempting to perform key import will result in an exception (e.g. invalid attribute. attribute should be set for User KEK.)

Option 1: User KEK with extractable and trusted. *Required for customers that need key import to be operational in addition to JCE, OpenSSL and other utilities to be operational. The example below we're setting -l (key label) as userkek, -t (key type) and -s (key size) as AES, 32 bytes.*

```
./azcloudhsm_util
loginHSM -u CU -s cu1 -p user1234
genSymKey -l userkek -t 31 -s 32 -wrap_with_trusted 1
```

```
Command:  genSymKey -l userkek -t 31 -s 32 -wrap_with_trusted 1

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 262259

        Cluster Status:
        Node id 1 status: 0x00000000 : HSM Return: SUCCESS
        Node id 2 status: 0x00000000 : HSM Return: SUCCESS
        Node id 3 status: 0x00000000 : HSM Return: SUCCESS
```

Option 2: User KEK with non-extractable but trusted. *Required for customers that do not need key import but require JCE, OpenSSL and other utilities to be operational. The example below we're setting -l (key label) as userkek, -t (key type) and -s (key size) as AES, 32 bytes and setting the key as non-extractable.*

./azcloudhsm_util
loginHSM -u CU -s cu1 -p user1234
genSymKey -l userkek -t 31 -s 32 -nex

```
Command:  loginHSM -u CU -s cu1 -p user1234

        Cfm3LoginHSM returned: 0x00 : HSM Return: SUCCESS

        Cluster Status:
        Node id 1 status: 0x00000000 : HSM Return: SUCCESS
        Node id 2 status: 0x00000000 : HSM Return: SUCCESS
        Node id 3 status: 0x00000000 : HSM Return: SUCCESS

Command:  genSymKey -l userkek -t 31 -s 32 -nex

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 262150

        Cluster Status:
        Node id 1 status: 0x00000000 : HSM Return: SUCCESS
        Node id 2 status: 0x00000000 : HSM Return: SUCCESS
        Node id 3 status: 0x00000000 : HSM Return: SUCCESS
```

2. After the key has been generated, customers will need to set the correct attributes on their key so that it can be used as a KEK. Firstly, you will need to end your azcloudhsm_util session. Customers will then run the management util and login as the Crypto Officer. You must be logged in as the Crypto Officer when setting the attributes on the Key.

```
sudo ./azcloudhsm_mgmt_util ./azcloudhsm_resource.cfg
loginHSM CO admin adminpassword
```

3. Upon assuming the role of the Crypto Officer and logging in, proceed to configure the attributes of the previously generated key. Obtain the key handle from the previous step and execute the following command to establish its attributes, utilizing your KeyHandleID.

Usage: setAttribute <KeyHandle> <AttributeID> <AttributeValue>. AttributeID 134 sets OBJ_ATTR_TRUSTED. AttributeValue 1 sets OBJ_ATTR_AUTH_FACTOR which is 1FA. Customers must use 134 1. No other values are supported as we only support 1FA.

```
setAttribute <KeyHandleId> 134 1
```

```
cloudmgmt>setAttribute 262150 134 1
**************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. Cav server does NOT synchronize these changes with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
******************************************************************

Do you want to continue(y/n)?y
setAttribute success on server 0(10.0.2.4)
setAttribute success on server 1(10.0.2.5)
setAttribute success on server 2(10.0.2.6)
cloudmgmt>
```

4. After configuring the attributes for the generated key, you can utilize it as a KEK (Key Encryption Key) by modifying the USER_KEK_HANDLE in your azcloudhsm_application.cfg file with the corresponding KeyHandleID.

```
DAEMON_ID=1
SOCKET_TYPE=UNIXSOCKET
PORT=1111
USER_KEK_HANDLE=262150
```

Verify OpenSSL installation and configuration with Azure Cloud HSM

To verify OpenSSL installation and configuration with Azure Cloud HSM we are going to perform an encrypt and decrypt operation using the Azure Cloud HSM OpenSSL engine.

Important Note: For your OpenSSL application to be successfully executed the azcloudhsm_application.cfg file must exist in the same location as your OpenSSL application and the azcloudhsm_client must be running. If the azcloudhsm_client is not running, you will receive a failed socket connection.

You may need to copy the azcloudhsm_application.cfg file from the default location below to the location of the OpenSSL application you are running. Follow the steps below to validate OpenSSL is configured and operational with Azure Cloud HSM.

Validating OpenSSL Engine:

1. **Copy the application.cfg to your home directory.**

```
cd /opt/azurecloudhsm/bin
cp ./azcloudhsm_application.cfg ~/azcloudhsm_application.cfg
```

2. **Start the client daemon** if it is not running.
   It's recommended for production to run the client daemon as a service. If you installed the Azure Cloud HSM SDK using deb or rpm, the client is not configured automatically to run as a service. The SDK during installation includes a service unit file under /etc/systemd/system/azure-cloud-hsm.service. To enable *azcloudhsm_client* to run as a service you will need to use the predefined azure-cloud-hsm.service file. You will then need to reload the Systemd configuration and enable the service to ensure its continuously running. For details on how to configure the client daemon to run as a service please reference the Azure Cloud HSM onboarding guide.

   The following steps below demonstrate two different ways to manually run the client daemon for testing OpenSSL integration.

```
cd /opt/azurecloudhsm/bin
sudo ./azcloudhsm_client azcloudhsm_resource.cfg
```

   You may also choose as an option to run the client daemon in the background using the following command or run the client daemon as a service to ensure it is always running.

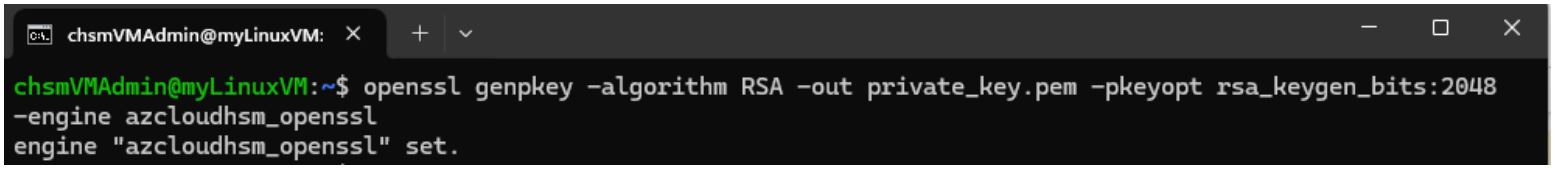
```
sudo ./azcloudhsm_client azcloudhsm_resource.cfg > /dev/null 2>&1 &
```

3. **Check Number of Existing Keys on HSM.** For a new Cloud HSM "total number of keys present" should be 1 as you created a user generated KEK prior to this step. If you have created other keys prior to this document, make note of the total number of keys so that you can validate the increment in the HSM upon next steps captured within this OpenSSL integration guide.

```
sudo ./azcloudhsm_util singlecmd loginHSM -u CU -s cu1 -p user1234 findKey
```

4. **Generate Private Key:** Use the following OpenSSL command to generate the private key on the HSM. Execute this command from your home directory, and do not run with sudo as it will run in a different session and fail to create the key.

```
cd ~
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048 -engine azcloudhsm_openssl
```

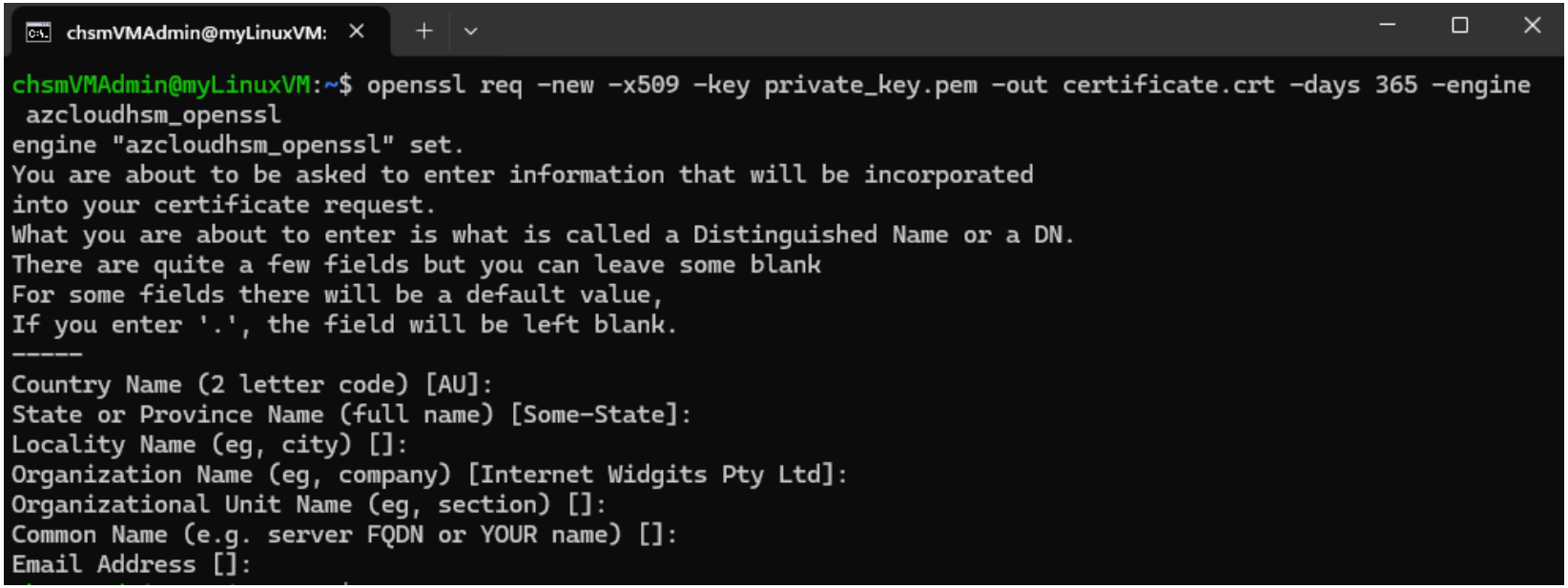


5. **Validate New Private Key Created:** Use the azcloudhsm_util to login and execute findKey to ensure private key was created. Change to /opt/azurecloudhsm/bin directory or use full path to azcloudhsm_util to validate private key was created. You should see the "total number of keys present" increment by one.

```
cd /opt/azurecloudhsm/bin
sudo ./azcloudhsm_util singlecmd loginHSM -u CU -s cu1 -p user1234 findKey
```

6. **Generate Digital Certificate:** Use the private key generated on the HSM to create a certificate using the following OpenSSL command. Run command from the home directory.

```
cd ~
openssl req -new -x509 -key private_key.pem -out certificate.crt -days 365 -engine azcloudhsm_openssl
```

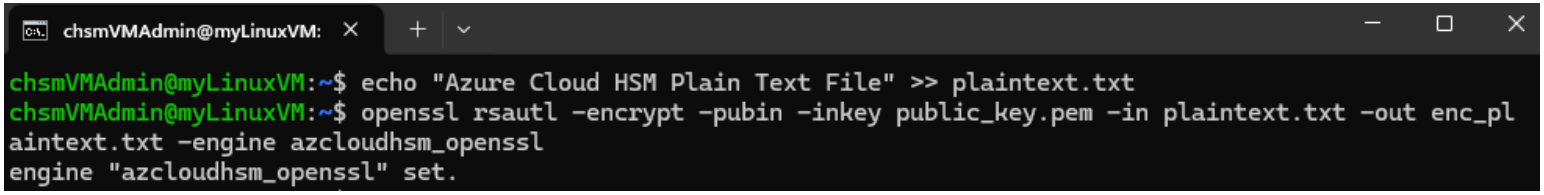


7. **Create a Public Key**

```
openssl rsa -in private_key.pem -pubout -out public_key.pem -engine azcloudhsm_openssl
```

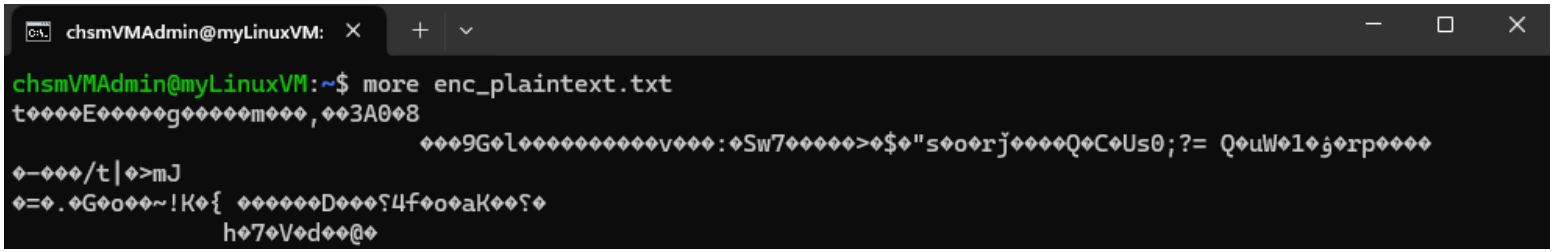
8. **Encrypt File.** In this example we're going to create a plain text file, sign that file and then output to an encrypted plaintext file.

```
echo "Azure Cloud HSM Plain Text File" >> plaintext.txt
openssl rsautl -encrypt -pubin -inkey public_key.pem -in plaintext.txt -out enc_plaintext.txt -engine azcloudhsm_openssl
```



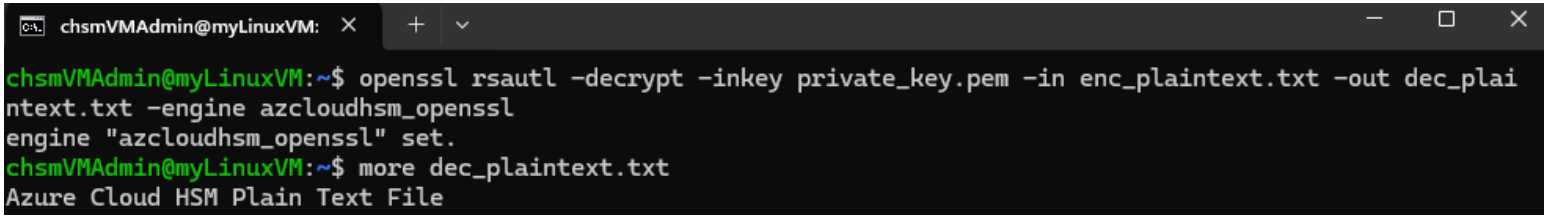
9. **Validate file is encrypted!**

```
more enc_plaintext.txt
```



10. **Decrypt File.** In this example we're going to decrypt the file we just created and validate the plain text.

```
openssl rsautl -decrypt -inkey private_key.pem -in enc_plaintext.txt -out dec_plaintext.txt -engine azcloudhsm_openssl
more dec_plaintext.txt
```



Configure Apache for SSL/TLS Offloading with Azure Cloud HSM

Important Note: In the provided instructions, the root user is utilized solely for illustrative purposes. Microsoft strongly advises employing a non-root user for improved management of user permissions.

STEP 1: Install Apache Web Server with support for SSL/TLS encryption.

After running these commands, Apache should be installed with SSL/TLS support on your Linux system. Keep in mind that additional configuration may be needed, such as setting up SSL certificates, depending on your specific requirements.

- **For Red Hat based (using yum):**

```
sudo yum update
sudo yum install httpd
sudo yum install mod_ssl
sudo systemctl start httpd
sudo systemctl enable httpd
```

- **For Red Hat based systems (using dnf):**

```
sudo dnf update
sudo dnf install httpd
sudo dnf install mod_ssl
sudo systemctl start httpd
sudo systemctl enable httpd
```

- **For Ubuntu based systems (using apt):**

```
sudo apt update
sudo apt install apache2
sudo a2enmod ssl
sudo systemctl start apache2
sudo systemctl enable apache2
```

STEP 2: Generate an RSA or EC Key Pair

Generate an RSA or EC key pair using the Azure Cloud HSM OpenSSL engine or

- **Option 1: Generate an RSA Key Pair**

```
openssl genrsa -engine azcloudhsm_openssl -out web_server_fake_PEM.key 2048
```

- **Option 2: Generate an EC Key Pair**

```
./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 genECCKeyPair -i 2 -l labelECApache
./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 getCaviumPrivKey -k {PRIVATE_KEY_HANDLE} -out
web_server_fake_PEM.key
```

STEP 3: Review the private key file contents.

Review contents of Private Key file which is in fake PEM format.

```
cat web_server_fake_PEM.key
```

```
chsmVMAdmin@myLinuxVM:~$ cat web_server_fake_PEM.key
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEICA/sLHQlra4WNBasNEV0lc5IlLsPoZXCQLbm9YCmi69oAoGCCqGSM49
AwEHoUQDQgAEAcsJX5ChN/iRLfasLR6l7+fzcrwlZcAGrNa7R4YuHh3hdB3Xf0hX
Rluo31Xgyb5NV5gGyFJex5T5H4yyYMvh/A==
-----END EC PRIVATE KEY-----
```

STEP 4: Generate the CSR

The command below generates a new CSR using the private key from "web_server_fake_PEM.key," and the resulting CSR is saved to a file named "web_server.csr." Additionally, it specifies "azcloudhsm_openssl" which is the Azure Cloud HSM OpenSSL engine.

```
openssl req -new -key web_server_fake_PEM.key -out web_server.csr -engine azcloudhsm_openssl
```

```
chsmVMAdmin@myLinuxVM:~$ openssl req -new -key web_server_fake_PEM.key -out web_server.csr -engine azcloudhsm_openssl
engine "azcloudhsm_openssl" set.
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

STEP 5: Review the CSR contents.

Review contents of CSR file.

```
cat web_server.csr
```

```
chsmVMAdmin@myLinuxVM:~$ cat web_server.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBADCBpwIBADBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEh
MB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMFkwEwYHKoZIzj0CAQYI
KoZIzj0DAQcDQgAEAcsJX5ChN/iRLfasLR6l7+fzcrwlZcAGrNa7R4YuHh3hdB3X
f0hXRluo31Xgyb5NV5gGyFJex5T5H4yyYMvh/KAAMAoGCCqGSM49BAMCA0gAMEUC
IEz11P8keg0iI2c5vKpg6LdpWOX+36UXpaFm9YUcKVvFAiEAlPKXPD7Az4aH0jP5
Hvpq8MG5EsritFpEIfD7/WRCOPc=
-----END CERTIFICATE REQUEST-----
```

STEP 6: Create a Self-Signed Certificate

The command below takes a CSR ("web_server.csr"), signs it using the private key from "web_server_fake.PEM.key," sets a validity period of 365 days, and outputs the signed certificate to a file named "web_server.crt." The "azcloudhsm_openssl" engine is explicitly specified for cryptographic operations.

```
openssl x509 -engine azcloudhsm_openssl -req -days 365 -in web_server.csr -signkey web_server_fake_PEM.key -out web_server.crt
```

```
chsmVMAdmin@myLinuxVM:~$ openssl x509 -engine azcloudhsm_openssl -req -days 365 -in web_server.csr -signkey web_server_
fake_PEM.key -out web_server.crt
engine "azcloudhsm_openssl" set.
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
Getting Private key
```

STEP 7: Enable Ports on the Apache Web Server

Ensure Ports for your Azure VM are enabled for HTTP/HTTPS. If you are running firewall on your Linux system you may need to execute the following commands.

```
sudo ufw allow 80
sudo ufw allow 443
sudo ufw status
```

STEP 8: Copy the Encoded Private Key Handle File and Webserver Certificate to required location.

The /etc/pki/tls directory is more commonly associated with Red Hat-based Linux distributions, such as Red Hat Enterprise Linux (RHEL). On Ubuntu and other Debian-based distributions, the equivalent directory for TLS-related files is typically /etc/ssl. Common subdirectories within /etc/pki/tls or /etc/ssl depending on your Linux distribution include certs for certificates, private for private keys, and private/certs for CA certificate.

RHEL:

```
sudo cp web_server_fake_PEM.key /etc/pki/tls/private/web_server_fake_PEM.key
sudo cp Webserver.crt /etc/pki/tls/certs/webserver.crt
```

Ubuntu:

```
sudo cp web_server_fake_PEM.key /etc/ssl/private/web_server_fake_PEM.key
sudo cp web_server.crt /etc/ssl/certs/webserver.crt
```

STEP 9: Change ownership of Encoded Private Key Handle File and Webserver Certificate

In the provided instructions, the following commands below are run to set file permissions to read, write, and execute for the owner, group, and others. In numeric mode, 7 corresponds to read (4) + write (2) + execute (1). Setting all three permissions results in 777. This is utilized solely for illustrative purposes and ease of setup demonstration.

Important Note: Setting file permission to 777 is very permissive and can be a security risk. Allowing anyone to write or execute a private key (localhost.key) is particularly dangerous, as it can compromise the security of the key. For private key files, more restrictive permissions (e.g. chmod 600) are often used to ensure that only the owner can read and write the key. If you are making these changes for a specific reason, make sure you fully understand the security implications, and consider more secure permission settings based on your use case and security requirements.

RHEL:

```
sudo chmod 777 /etc/pki/tls/private/web_server_fake_PEM.key
sudo chmod 777 /etc/pki/tls/certs/webserver.crt
```

Ubuntu:

```
sudo chmod 777 /etc/ssl/private/web_server_fake_PEM.key
sudo chmod 777 /etc/ssl/certs/webserver.crt
```

STEP 10: Update the server certificate and server encoded private key handle file location.

To update the server certificate and server private key location for Apache, you'll need to modify the Apache configuration file. The specific file you need to edit might vary depending on your Apache version, configuration, and Linux distribution. You will need to restart the Apache services after making changes to the configuration. Additionally, you may need to update the Cipher Suite in the same configuration file.

Important Note: Regarding SSLCryptoDevice directive in Apache configuration file. The SSLCryptoDevice directive is no longer used to enable hardware-based SSL acceleration in Apache. Instead, Apache typically relies on external libraries like OpenSSL for SSL/TLS operations. Since you are using a hardware security module (Azure Cloud HSM), the integration is often handled through the OpenSSL library and its configuration.

Important Note: For SSLCipherSuite insert the following as a single line without any line breaks. Failure to do so may result in an error when enabling mod_ssl.

RHEL:

```
cd /etc/httpd/conf.d
sudo vim ssl.conf
sudo systemctl restart httpd
```

Ubuntu:

```
cd /etc/apache2/sites-available
sudo vim default-ssl.conf
sudo systemctl restart apache2
```

Conf File:

```
SSLEngine    on
SSLCertificateFile    /path/to/your/webserver.crt
SSLCertificateKeyFile    /path/to/your/web_server_fake_PEM.key

SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

STEP 11: Configure an environment-values file as needed.

The following command will open the specified service file in the Vim text editor. The specific file you need to edit might vary depending on your Apache version, configuration, and Linux distribution. You will need to restart the Apache services after making changes to the configuration.

RHEL:

```
sudo vim /lib/systemd/system/httpd.service
sudo systemctl restart httpd
```

Ubuntu:

```
sudo vim /lib/systemd/system/apache2.service
sudo systemctl restart apache2
```

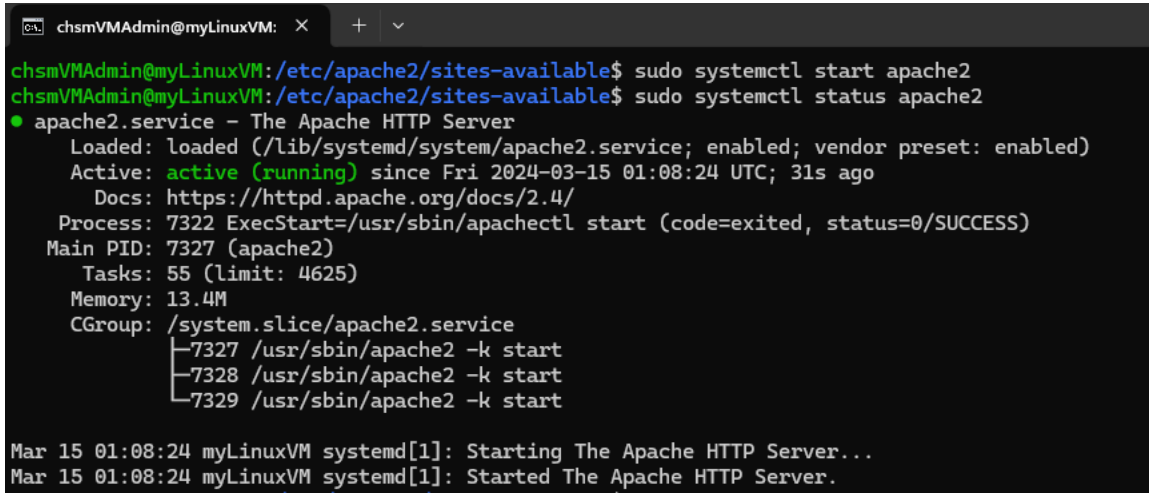
STEP 12: Start the HTTP Service

RHEL:

```
sudo service httpd start
sudo service httpd status
```

Ubuntu:

```
sudo systemctl start apache2
sudo systemctl status apache2
```



STEP 13: Verify that the SSL/TLS handshake is being offloaded to the Azure Cloud HSM.

The following can be done using OpenSSL or a Web Brower. In the provided instructions we will use OpenSSL s_client (e.g. openssl s_client -connect 10.0.2.7:443). In the command output, verify that the certificate matches.

```
openssl s_client -connect {IPADDRESS:PORT}
```

```
chsmVMAdmin@myLinuxVM:/etc/apache2/sites-available$ openssl s_client -connect 10.0.2.7:443
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
verify error:num=18:self signed certificate
verify return:1
depth=0 C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
verify return:1
---
Certificate chain
 0 s:C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
   i:C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIBhDCCASsCFE18xaj6nEoat1k2a+AeNMt/BCCjMAoGCCqGSM49BAMCMEUxCzAJ
BgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5l
dCBXaWRnaXRzIFB0eSBMdGQwHhcNMjQwMzE1MDEwMDM5WhcNMjUwMzE1MDEwMDM5
WjBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwY
SW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcD
QgAEAcsJX5ChN/iRLfasLR6l7+fzcrwlZcAGrNa7R4YuHh3hdB3Xf0hXRluo31Xg
yb5NV5gGyFJex5T5H4yyYMvh/DAKBggqhkjOPQQDAgNHADBEAiA9ZDf5wcoLA5w7
eXjpDT4Ly3udZ5PPyBeEN+CTgWrQqwIgL1zWJAjHALCj2XkugSjiVhus3SIXLPN/
Xin1RNkOjno=
-----END CERTIFICATE-----
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd

issuer=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: ECDSA
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 763 bytes and written 363 bytes
Verification error: self signed certificate
---
```

```
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 256 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 18 (self signed certificate)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: 6EA0216766B565D8CF51BAD325C2313793BFA37FFDDAAD1883AAFC830653E79F
    Session-ID-ctx:
    Resumption PSK: F2712972CF8D4F6DC14314EA739FCED0082B3D7C37CBFE45C45E2C19B9EA5E2575D1895CAD863A899882C8EC0EA1EF9A
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 0f be 0c 24 5d 7e 8d 9d-de ae ae 36 00 45 b5 11   ...$]~.....6.E..
    0010 - c4 e6 02 8c af be d3 34-ae fa a6 13 fb 5d 58 54   .......4.....]XT
    0020 - 2c 76 76 69 51 d7 72 26-91 69 f5 e8 66 76 3f 2a   ,vviQ.r&.i..fv?*
    0030 - eb e2 ca 4a 31 1b 2d cf-eb e2 33 b4 9e ac dc be   ...J1.-...3.....
    0040 - 19 e6 4d 7a 9a 97 12 33-fe f9 fc dc 07 89 c0 38   ..Mz...3.......8
    0050 - d7 d3 2b 24 ec 63 26 3a-c1 9c 88 2b 25 20 57 59   ..+$.c&:...+% WY
    0060 - cb 42 4f d5 29 85 65 db-7e 35 77 3b e8 27 78 54   .BO.).e.~5w;.'xT
    0070 - 3f ec c7 c5 1c aa e6 53-37 e1 fa e4 11 5e 94 ff   ?......S7....^..
    0080 - 3b 09 a1 83 40 37 23 4a-ac 03 59 aa 4f f3 f3 71   ;...@7#J..Y.O..q
    0090 - 43 5c 42 ae b6 d4 40 42-fb f7 6b 63 68 0e 1e d3   C\B...@B..kch...
    00a0 - 3a 57 a8 2e 2f 9f 81 77-ac 26 cb 0c 5e ac 1c c6   :W../..w.&..^...
    00b0 - e2 b6 2f aa ed d1 5c a6-db af 00 f9 d3 7a d6 8b   ../...\......z..
    00c0 - de 3c 73 d1 e1 b1 c0 61-fd d3 55 75 43 35 d6 2b   .<s....a..UuC5.+
    00d0 - 14 f9 d1 a9 fe bb e1 a0-df 6b b7 ec 2a b9 e4 4a   .........k..*..J

    Start Time: 1710475022
```

```
    Timeout   : 7200 (sec)
    Verify return code: 18 (self signed certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: 8F58CB21EBB4EEDA8C7D89263645E68A349CCF54E07317FB3A3E80BA6D80566D
    Session-ID-ctx:
    Resumption PSK: DAAC59B333078C411CE0CC7446079FC56F107FF0D836F8C8D16E488A1179075E3EDCEF3959AE2E41185A8AB8C60A16DB
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 0f be 0c 24 5d 7e 8d 9d-de ae ae 36 00 45 b5 11   ...$]~.....6.E..
    0010 - 2b 7a ad 55 5d 1a 25 16-4e e7 5d df fc 42 a0 21   +z.U].%.N.]..B.!
    0020 - 29 af 70 05 41 54 a3 a6-6a da 4a 60 ba 27 73 f3   ).p.AT..j.J`.'s.
    0030 - 1c 9d 3e ea 08 eb 66 89-77 50 c4 33 de da 66 88   ..>...f.wP.3..f.
    0040 - da e8 32 20 cd d5 a1 e1-cc e2 17 7b 12 31 1a 1a   ..2 .......{.1..
    0050 - d2 c2 d9 8c 99 6d b5 67-b6 05 5d e7 8c c4 c0 5d   .....m.g..]....]
    0060 - c0 f0 aa b5 7d a1 99 f1-76 0f f4 b6 31 0b 92 cc   ....}...v...1...
    0070 - 66 44 9a 21 fe 1d a8 48-c9 a0 26 66 ea e9 41 5f   fD.!...H..&f..A_
    0080 - 03 5a 8d 40 cf 6f 0c d7-61 d2 91 1c 36 21 c7 c0   .Z.@.o..a...6!..
    0090 - b7 11 99 d2 4c 8e ca ec-0a 35 8b d0 46 c5 3f 92   ....L....5..F.?.
    00a0 - 9e 83 89 fd f8 e2 ce 3e-5a 68 48 c3 c5 da f1 53   .......>ZhH....S
    00b0 - 63 27 6b a3 44 67 ef c6-35 72 e0 ba 75 3b 47 68   c'k.Dg..5r..u;Gh
    00c0 - 80 34 bd 28 a2 aa f9 42-ba 3c 1a 24 2c 05 68 09   .4.(...B.<.$,.h.
    00d0 - bc bf 23 c1 10 b3 eb 2e-7a 86 a6 d9 e6 b9 22 ba   ..#.....z.....".

    Start Time: 1710475022
    Timeout   : 7200 (sec)
    Verify return code: 18 (self signed certificate)
    Extended master secret: no
```

```
    Max Early Data: 0
---
read R BLOCK
closed
```

Configure Nginx for SSL/TLS Offloading with Azure Cloud HSM

Important Note: In the provided instructions, the root user is utilized solely for illustrative purposes. Microsoft strongly advises employing a non-root user for improved management of user permissions.

STEP 1: Install Nginx with support for SSL/TLS encryption.

After running these commands, Nginx should be installed with SSL/TLS support on your Linux system. Keep in mind that additional configuration may be needed, such as setting up SSL certificates, depending on your specific requirements.

- **For Red Hat based (using yum):**

```
sudo yum update
sudo yum install nginx
sudo systemctl start nginx
sudo systemctl enable nginx
```

- **For Red Hat based systems (using dnf):**

```
sudo dnf update
sudo dnf install epel-release
sudo dnf install nginx
sudo systemctl start nginx
sudo systemctl enable nginx
```

- **For Ubuntu based systems (using apt):**

```
sudo apt update
sudo apt install nginx
sudo systemctl start nginx
sudo systemctl enable nginx
```

STEP 2: Generate an RSA or EC Key Pair

Generate an RSA or EC key pair using the Azure Cloud HSM OpenSSL engine.

- **Option 1: Generate an RSA Key Pair**

```
openssl genrsa -engine azcloudhsm_openssl -out web_server_fake_PEM.key 2048
```

- **Option 2: Generate an EC Key Pair**

```
./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 genECCKeyPair -i 2 -l labelECNginx
./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 getCaviumPrivKey -k {PRIVATE_KEY_HANDLE} -out web_server_fake_PEM.key
```

STEP 3: Review the private key file contents.

Review contents of Private Key file which is in fake PEM format.

```
cat web_server_fake_PEM.key
```

```
chsmVMAdmin@myLinuxVM:~$ cat web_server_fake_PEM.key
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEINCShh0QMjeYKpLHREF+kLzbGvdYkucvMQ3gh5hLvITyoAoGCCqGSM49
AwEHoUQDQgAErh6CWio6IVdBV5PMvZC9rYuzOLckd/FhMbmDHlj48St528nVWRw7
s0FVumo0AXT6lITnWLT482n+cI+hIbBxZg==
-----END EC PRIVATE KEY-----
```

STEP 4: Generate the CSR

The command below generates a new CSR using the private key from "web_server_fake_PEM.key," and the resulting CSR is saved to a file named "web_server.csr." Additionally, it specifies "azcloudhsm_openssl" which is the Azure Cloud HSM OpenSSL engine.

```
openssl req -new -key web_server_fake_PEM.key -out web_server.csr -engine azcloudhsm_openssl
```

```
chsmVMAdmin@myLinuxVM:~$ openssl req -new -key web_server_fake_PEM.key -out web_server.csr -engine azcloudhsm_openssl
engine "azcloudhsm_openssl" set.
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

STEP 5: Review the CSR contents.

Review contents of CSR file.

```
cat web_server.csr
```

```
chsmVMAdmin@myLinuxVM:~$ cat web_server.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBATCBpwIBADBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEh
MB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMFkwEwYHKoZIzj0CAQYI
KoZIzj0DAQcDQgAErh6CWio6IVdBV5PMvZC9rYuzOLckd/FhMbmDHlj48St528nV
WRw7s0FVumo0AXT6lITnWLT482n+cI+hIbBxZqAAMAoGCCqGSM49BAMCA0kAMEYC
IQDy4A+5Qq1K0JV7Hx64xLb2klbtXIcfOc2mYGXDM+niDQIhANQueQMFzOzlbYOt
1qK4ZoY93U1VnYAcQJCHt4R9VZfs
-----END CERTIFICATE REQUEST-----
```

STEP 6: Create a Self-Signed Certificate

The command below takes a CSR ("web_server.csr"), signs it using the private key from "web_server_fake.PEM.key," sets a validity period of 365 days, and outputs the signed certificate to a file named "web_server.crt." The "azcloudhsm_openssl" engine is explicitly specified for cryptographic operations.

```
openssl x509 -engine azcloudhsm_openssl -req -days 365 -in web_server.csr -signkey web_server_fake_PEM.key -out web_server.crt
```

```
chsmVMAdmin@myLinuxVM:~$ openssl x509 -engine azcloudhsm_openssl -req -days 365 -in web_server.csr -signkey web_server_fake_PEM.key
 -out web_server.crt
engine "azcloudhsm_openssl" set.
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
Getting Private key
```

STEP 7: Enable Ports on the Nginx Web Server

Ensure Ports for your Azure VM are enabled for HTTP/HTTPS. If you are running firewall on your Linux system you may need to execute the following commands.

```
sudo ufw allow 80
sudo ufw allow 443
sudo ufw status
```

STEP 8: Copy the Encoded Private Key Handle File and Webserver Certificate to required location.

The /etc/nginx is the default directory for majority of Linux distributions. Common subdirectories may include /etc/nginx/private, /etc/nginx/tls and /etc/nginx/ssl depending on your Linux distribution. Includes certs for certificates, private for private keys, and private/certs for CA certificate.

```
sudo mkdir /etc/nginx/private
sudo cp web_server_fake_PEM.key /etc/nginx/private
sudo cp web_server.crt /etc/nginx
```

STEP 9: Change ownership of Encoded Private Key Handle File and Webserver Certificate

In the provided instructions, the following commands below are run to set file permissions to read, write, and execute for the owner, group, and others. In numeric mode, 7 corresponds to read (4) + write (2) + execute (1). Setting all three permissions results in 777. This is utilized solely for illustrative purposes and ease of setup demonstration.

Important Note: Setting file permission to 777 is very permissive and can be a security risk. Allowing anyone to write or execute a private key (localhost.key) is particularly dangerous, as it can compromise the security of the key. For private key files, more restrictive permissions (e.g. chmod 600) are often used to ensure that only the owner can read and write the key. If you are making these changes for a specific reason, make sure you fully understand the security implications, and consider more secure permission settings based on your use case and security requirements.

```
sudo chmod 777 /etc/nginx/web_server.crt
sudo chmod 777 /etc/nginx/private/web_server_fake_PEM.key
```

STEP 10: Backup the Nginx conf file.

After running the following command, you'll have a backup copy of your original nginx.conf file with the name nginx.conf.backup.

```
sudo cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.backup
```

STEP 11: Update the server certificate and server encoded private key handle file location.

To update the server certificate and server private key location for Nginx, you'll need to modify the Nginx configuration file. The specific file you need to edit might vary depending on your Nginx version, configuration, and Linux distribution. You will need to restart the Nginx services after making changes to the configuration. Additionally, you may need to update the Cipher Suite in the same configuration file.

Important Note: For SSLCipherSuite insert the following as a single line without any line breaks. Failure to do so may result in an error when enabling mod_ssl.

```
cd /etc/nginx/sites-available
sudo vim default
sudo systemctl restart nginx
```

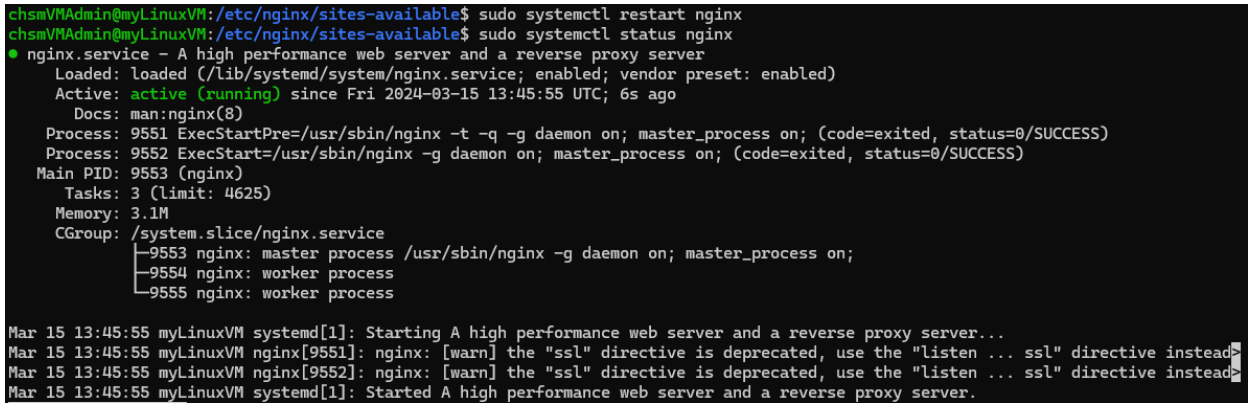
Conf File:

```
server {
   listen 443 ssl;
   server_name localhost;
   ssl on;
   ssl_certificate /etc/nginx/web_server.crt;
   ssl_certificate_key /etc/nginx/private/web_server_fake_PEM.key;
   ssl_session_timeout 5m;
   ssl_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2;
   ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-GCM-
SHA384:TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!SRP:!CAMELLIA:HIGH:!aNULL:
!MD5;
```

```
    # Your other server configuration
    location / {
        # Your other directives
        root html;
        index index.html index.htm;
    }
}
```

STEP 12: Check status that Nginx Service is running.

```
sudo systemctl status nginx
```



STEP 13: Verify that the SSL/TLS handshake is being offloaded to the Azure Cloud HSM.

The following can be done using OpenSSL or a Web Brower. In the provided instructions we will use OpenSSL s_client (e.g. openssl s_client -connect 10.0.2.7:443). In the command output, verify that the certificate matches.

```
openssl s_client -connect {IPADDRESS:PORT}
```

```
chsmVMAdmin@myLinuxVM:/etc/nginx/sites-available$ openssl s_client -connect 10.0.2.7:443
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
verify error:num=18:self signed certificate
```

```
verify return:1
depth=0 C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
verify return:1
---
Certificate chain
 0 s:C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
   i:C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIBhDCCASsCFCrepNbJeqpGripPZpNWvTUR56MXMAoGCCqGSM49BAMCMEUxCzAJ
BgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5l
dCBXaWRnaXRzIFB0eSBMdGQwHhcNMjQwMzE1MTMzODAwWhcNMjUwMzE1MTMzODAw
WjBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwY
SW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcD
QgAErh6CWio6IVdBV5PMvZC9rYuzOLckd/FhMbmDHlj48St528nVWRw7s0FVumo0
AXT6llTnWLT482n+cI+hIbBxZjAKBggqhkjOPQQDAgNHADBEAiAbkmlcdDCnl8pk
wJV6S33V4J5m4ZxYqk58yd+cEnu+ewIgOUneWZkbgxofnib6v56U0v9T0wUXmAei
M9MY/O9P7D0=
-----END CERTIFICATE-----
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd

issuer=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: ECDSA
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 848 bytes and written 376 bytes
Verification error: self signed certificate
---
New, TLSv1.2, Cipher is ECDHE-ECDSA-AES256-GCM-SHA384
Server public key is 256 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
```

```
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-ECDSA-AES256-GCM-SHA384
    Session-ID: D5B178ECAD92437562332DD62F5E8DA411248851423FA8CE866CE2A3058E8B5A
    Session-ID-ctx:
    Master-Key: 57F674F35960CE0AB61652A1D59EFD6F36E6D21BD24A7D3AC8BA81DD1037F504CBB24FA7BE50C1D32CB8DB850695A1E5
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 16 3a be ba a1 61 be 67-c7 4e 8d 26 02 da 21 bf   .:...a.g.N.&..!.
    0010 - ad 6c 66 2e 2c 4f 50 1a-3e a2 55 e7 66 1d 48 c2   .lf.,OP.>.U.f.H.
    0020 - ae d1 03 70 97 c7 a6 29-82 aa 5b 38 7c 91 1f a0   ...p...)..[8|...
    0030 - 98 b8 7f ee 1d af d4 8e-e1 c1 b5 31 00 6e 48 f3   ...........1.nH.
    0040 - 4a 77 d3 6f f7 54 aa 9b-77 94 75 3c 6b 0d fa f0   Jw.o.T..w.u<k...
    0050 - f0 03 e8 31 15 ce c2 e5-6a d9 33 a1 bd 81 13 6b   ...1....j.3....k
    0060 - 3d b5 d9 1b 52 97 3a 9c-00 6a 80 8f cb de 4b d5   =...R.:..j....K.
    0070 - 7a b8 fa 84 81 2c 98 15-9e a7 6b f0 41 f2 3f 76   z....,....k.A.?v
    0080 - 78 91 31 c5 24 3a 86 f2-90 cb a9 af 9f 4e e0 b9   x.1.$:.......N..
    0090 - be 1a 79 f7 1e 29 98 7e-71 b1 64 cd e2 b1 07 05   ..y..).~q.d.....
    00a0 - fe 33 e7 65 6d ca f4 ad-56 e6 3a 92 2b e4 38 ef   .3.em...V.:.+.8.

    Start Time: 1710510424
    Timeout   : 7200 (sec)
    Verify return code: 18 (self signed certificate)
    Extended master secret: yes
---
read:errno=0
```

Download and Compile Nginx from source for SSL/TLS Offloading with Azure Cloud HSM

Important Note: In the provided instructions, the root user is utilized solely for illustrative purposes. Microsoft strongly advises employing a non-root user for improved management of user permissions.

STEP 1: Download and untar the Nginx source package in the /home directory.

After running these commands, you will have a directory named nginx-* containing the NGINX source code. You can navigate into this directory to configure, compile, and install NGINX on your system.

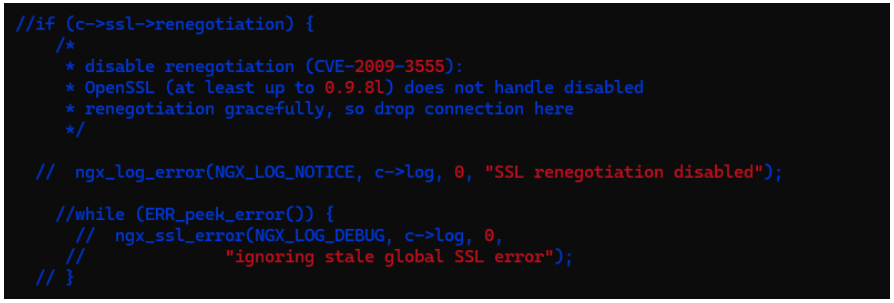
```
cd /home
sudo wget http://www.nginx.org/download/nginx-1.18.0.tar.gz
sudo tar -zxvf nginx-1.18.0.tar.gz
```

STEP 2: Enable renegotiation in Nginx.

Important Note: If configuring Nginx from source package, by default, renegotiation is disabled. To enable it, open the ngx_event_openssl.c file in a text editor and make the following changes:

```
sudo vim /home/nginx-*/src/event/ngx_event_openssl.c
```

- Find :/ssl->renegotiation and comment out the entire if condition within the ngx_event_openssl.c file.
  // if (c->ssl->renegotiation) {
  # // .. … … … … .. . #
  // .. … … … … .. . #
  // }



```
//if (c->ssl->renegotiation) {
    /*
     * disable renegotiation (CVE-2009-3555):
     * OpenSSL (at least up to 0.9.8l) does not handle disabled
     * renegotiation gracefully, so drop connection here
     */

  //  ngx_log_error(NGX_LOG_NOTICE, c->log, 0, "SSL renegotiation disabled");

    //while (ERR_peek_error()) {
      //  ngx_ssl_error(NGX_LOG_DEBUG, c->log, 0,
      //                "ignoring stale global SSL error");
  // }
```

- Find :/SSL3_FLAGS_NO_RENEGOTIATE_CIPHERS and comment out within the ngx_event_openssl.c file.
  //c->ssl->connection->s3->flags |= SSL3_FLAGS_NO_RENEGOTIATE_CIPHERS;

```
/* initial handshake done, disable renegotiation (CVE-2009-3555) */
if (c->ssl->connection->s3) {
    //c->ssl->connection->s3->flags |= SSL3_FLAGS_NO_RENEGOTIATE_CIPHERS;
}
```

STEP 3: Configure the Nginx sources.

The following options are specific to the NGINX build process and are used to customize the features and dependencies of the resulting executable. The actual options might vary based on the NGINX version and the system's configuration.

```
cd /home/nginx-*/
sudo ./configure --prefix=/etc/nginx --with-ld-opt=-pthread --with-http_ssl_module --without-pcre --without-http_rewrite_module --with-cc-opt=-DOPENSSL_NO_NEXTPROTONEG
```

STEP 4: Compile and Install Nginx

After NGINX installation completes, the location is under /usr/local/nginx.

```
sudo make
sudo make install
```

STEP 5: Validate Nginx installed.

The following command checks the version of Nginx installed on your system.

```
/etc/nginx/sbin/nginx -v
```

STEP 6: Generate an RSA or EC Key Pair

Generate an RSA or EC key pair using the Azure Cloud HSM OpenSSL engine.

- **Option 1: Generate an RSA Key Pair**

  ```
  openssl genrsa -engine azcloudhsm_openssl -out web_server_fake_PEM.key 2048
  ```

- **Option 2: Generate an EC Key Pair**

  ```
  ./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 genECCKeyPair -i 2 -l labelECNginx
  ./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 getCaviumPrivKey -k {PRIVATE_KEY_HANDLE} -out web_server_fake_PEM.key
  ```

STEP 7: Review the private key file contents.

Review contents of Private Key file which is in fake PEM format.

```
cat web_server_fake_PEM.key
```

```
chsmVMAdmin@myLinuxVM:~$ cat web_server_fake_PEM.key
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIBx8RmoxoJFJayqy7eb2VNKx8Gr4lOWa+H+aDj9t93vJoAoGCCqGSM49
AwEHoUQDQgAEeSmb4/wdyrXkJ20VYuCeLHdbEPnTiV6+7l76QDeOpFFap+FjsPF/
2sSxGQKQ2h+30/8I6+c5Avv/4/7IUarIHA==
-----END EC PRIVATE KEY-----
```

STEP 8: Generate the CSR

The command below generates a new CSR using the private key from "web_server_fake_PEM.key," and the resulting CSR is saved to a file named "web_server.csr." Additionally, it specifies "azcloudhsm_openssl" which is the Azure Cloud HSM OpenSSL engine.

```
openssl req -new -key web_server_fake_PEM.key -out web_server.csr -engine azcloudhsm_openssl
```

```
chsmVMAdmin@myLinuxVM:~$ openssl req -new -key web_server_fake_PEM.key -out web_server.csr -engine azcloudhsm_openssl
engine "azcloudhsm_openssl" set.
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

STEP 9: Review the CSR contents.

Review contents of CSR file.

```
cat web_server.csr
```

```
chsmVMAdmin@myLinuxVM:~$ cat web_server.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBADCBpwIBADBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEh
MB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMFkwEwYHKoZIzj0CAQYI
KoZIzj0DAQcDQgAEeSmb4/wdyrXkJ20VYuCeLHdbEPnTiV6+7l76QDeOpFFap+Fj
sPF/2sSxGQKQ2h+30/8I6+c5Avv/4/7IUarIHKAAMAoGCCqGSM49BAMCA0gAMEUC
IENes5V0Fr8VsR05pjp5AlwdfaP9JRyh+A89pHM/82/gAiEAsrr1V+74LYrO7jyR
FDublJfgR85BNY2ECoPm/1ircv4=
-----END CERTIFICATE REQUEST-----
```

STEP 10: Create a Self-Signed Certificate

The command below takes a CSR ("web_server.csr"), signs it using the private key from "web_server_fake.PEM.key," sets a validity period of 365 days, and outputs the signed certificate to a file named "web_server.crt." The "azcloudhsm_openssl" engine is explicitly specified for cryptographic operations.

```
openssl x509 -engine azcloudhsm_openssl -req -days 365 -in web_server.csr -signkey web_server_fake_PEM.key -out web_server.crt
```

```
chsmVMAdmin@myLinuxVM:~$ openssl x509 -engine azcloudhsm_openssl -req -days 365 -in web_server.csr -signkey web_server_fake_PEM.key
 -out web_server.crt
engine "azcloudhsm_openssl" set.
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
Getting Private key
```

STEP 11: Enable Ports on the Nginx Web Server

Ensure Ports for your Azure VM are enabled for HTTP/HTTPS. If you are running firewall on your Linux system you may need to execute the following commands.

```
sudo ufw allow 80
sudo ufw allow 443
sudo ufw status
```

STEP 12: Copy the Encoded Private Key Handle File and Webserver Certificate to required location.

The /etc/nginx is the default directory for majority of Linux distributions. Common subdirectories may include /etc/nginx/private, /etc/nginx/tls and /etc/nginx/ssl depending on your Linux distribution. Includes certs for certificates, private for private keys, and private/certs for CA certificate.

```
sudo mkdir /etc/nginx/private
sudo cp web_server_fake_PEM.key /etc/nginx/private
sudo cp web_server.crt /etc/nginx
```

STEP 13: Change ownership of Encoded Private Key Handle File and Webserver Certificate

In the provided instructions, the following commands below are run to set file permissions to read, write, and execute for the owner, group, and others. In numeric mode, 7 corresponds to read (4) + write (2) + execute (1). Setting all three permissions results in 777. This is utilized solely for illustrative purposes and ease of setup demonstration.

Important Note: Setting file permission to 777 is very permissive and can be a security risk. Allowing anyone to write or execute a private key (localhost.key) is particularly dangerous, as it can compromise the security of the key. For private key files, more restrictive permissions (e.g. chmod 600) are often used to ensure that only the owner can read and write the key. If you are making these changes for a specific reason, make sure you fully understand the security implications, and consider more secure permission settings based on your use case and security requirements.

```
sudo chmod 777 /etc/nginx/web_server.crt
sudo chmod 777 /etc/nginx/private/web_server_fake_PEM.key
```

STEP 14: Backup the Nginx conf file.

After running the following command, you'll have a backup copy of your original nginx.conf file with the name nginx.conf.backup.

```
sudo cp /etc/nginx/conf/nginx.conf /etc/nginx/conf/nginx.conf.backup
```

STEP 15: Update the server certificate and server encoded private key handle file location.

To update the server certificate and server private key location for Nginx, you'll need to modify the Nginx configuration file. The specific file you need to edit might vary depending on your Nginx version, configuration, and Linux distribution. You will need to restart the Nginx services after making changes to the configuration. Additionally, you may need to update the Cipher Suite in the same configuration file.

Important Note: For SSLCipherSuite insert the following as a single line without any line breaks. Failure to do so may result in an error when enabling mod_ssl.

```
sudo vim /etc/nginx/conf/nginx.conf
```

Conf File:

```
server {
   listen 443 ssl;
   server_name localhost;
   ssl on;
   ssl_certificate /etc/nginx/web_server.crt;
   ssl_certificate_key /etc/nginx/private/web_server_fake_PEM.key;
   ssl_session_timeout 5m;
   ssl_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2;
   ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-GCM-
SHA384:TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!SRP:!CAMELLIA:HIGH:!aNULL:
!MD5;

   # Your other server configuration
   location / {
      # Your other directives
      root html;
      index index.html index.htm;
   }
}
```

STEP 16: Copy shared object file libazcloudhsm_openssl.so to /engines-*

Before initiating Nginx, transfer the shared object file 'libazcloudhsm_openssl.so' from the Azure Cloud HSM SDK directory to the designated destination. It is crucial to deliberately rename the file to 'libazcloudhsm_openssl.so' during this operation.

```
sudo cp /usr/local/lib64/AzureCloudHSM-ClientSDK-*/libazcloudhsm_openssl.so /usr/lib/x86_64-linux-gnu/engines-
1.1/libazcloudhsm_openssl.so
```

STEP 17: Run Nginx Server

Run Nginx server as root and pass in the environment variables with -E

```
cd /etc/nginx
sudo -E ./sbin/nginx -c conf/nginx.conf
```

STEP 18: Verify that the SSL/TLS handshake is being offloaded to the Azure Cloud HSM.

The following can be done using OpenSSL or a Web Brower. In the provided instructions we will use OpenSSL s_client (e.g. openssl s_client -connect 10.0.2.7:443). In the command output, verify that the certificate matches.

```
openssl s_client -connect {IPADDRESS:PORT}
```

Import your own key.

Important Note: You will need to reference your user generated KEK handle id to process key import.

STEP 1: Generate an RSA or EC key pair.

Generate an RSA or EC key pair using the Azure Cloud HSM OpenSSL engine.

- **Option 1: Generate an RSA Key Pair**

```
openssl genrsa  -out privateKey.pem 2048
```

- **Option 2: Generate an EC Key Pair**

```
./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 genECCKeyPair -i 2 -l labelECImport
./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 getCaviumPrivKey -k {PRIVATE_KEY_HANDLE} -out privateKey.pem
```

STEP 2: Generate the public key out of the private key.

The following command takes a private key from the file privateKey.pem, extracts the corresponding public key, and saves it to a new file named publicKey.pem in PEM format.

- **Option 1: Generate an RSA Key Pair**

```
openssl rsa -in privateKey.pem -outform PEM -pubout -out publicKey.pem
```

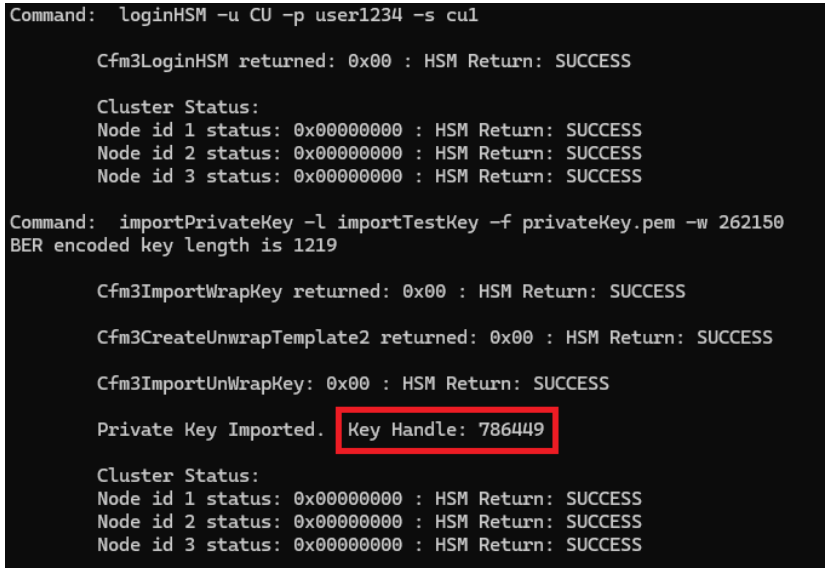
- **Option 2: Generate an EC Key Pair**

```
openssl ec -in privateKey.pem -outform PEM -pubout -out publicKey.pem
```

STEP 3: Import the private key into Azure Cloud HSM

The following command is instructing to import a private key from the file privateKey.pem into your Azure Cloud HSM, assigning it the label "importTestKey" and specifying a key wrapping mechanism with the parameter -w {UserGenerateKEKId).

```
./azcloudhsm_util
loginHSM -u CU -p user1234 -s cu1
importPrivateKey -l importTestKey -f privateKey.pem -w {UserGenerateKEKId}
```



STEP 4: Export the private key handle in fake PEM format and save it to a file.

The following command is instructing getCaviumPrivKey command to encode the key handle and save it to a file named fake_PEM.key in PEM format. The parameter '-k' specifies the RSA or ECDSA private key handle.

```
getCaviumPrivKey -k {PrivateKeyHandle} -out fake_PEM.key
```

```
Command:  getCaviumPrivKey -k 786449 -out fake_PEM.key

Private Key Handle is written to fake_PEM.key in fake PEM format

        getCaviumPrivKey returned: 0x00 : HSM Return: SUCCESS
```

Appendix

Frequently Asked Questions

- **Does Azure Cloud HSM OpenSSL Engine support Windows?**
  No. Azure Cloud HSM OpenSSL Engine supports Linux (Ubuntu 20.04, Ubuntu 22.04, RHEL 7, RHEL 8, CBL Mariner 2) only.

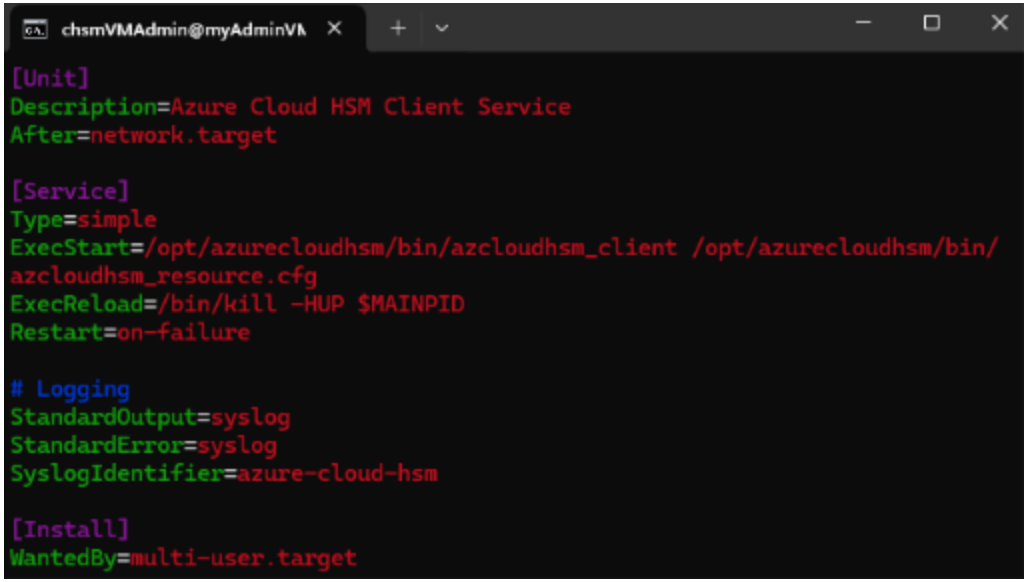
- **Does Azure Cloud HSM OpenSSL Engine support PKCS#11 for SSL/TLS offloading?**
  No. The requirement of OpenSSL supporting PKCS#11 for SSL/TLS offloading is not supported by Azure Cloud HSM. Customers that require PKCS#11 for SSL/TLS Offloading must use the TLS Offload Library for Azure Managed HSM.

- **Why am I getting error message Failed to connect socket, LIQUIDSECURITY: Daemon socket connection error when running azcloudhsm_util?**
  The error message suggests that azcloudhsm_client is not running. You need to ensure that the azcloudhsm_client is running for the Azure Cloud HSM utilities to properly execute. You can start the azcloudhsm_client by ensuring its running as a service or by manually running the following command in a separate terminal.

  Linux (Recommended):
  If you installed the Azure Cloud HSM SDK using deb or rpm, the client is not configured automatically to run as a service. The SDK during installation includes a service unit file under /etc/systemd/system/azure-cloud-hsm.service. To enable *azcloudhsm_client* to run as a service you will need to use the predefined azure-cloud-hsm.service file. You will then need to reload the Systemd configuration and enable the service to ensure its continuously running by performing the following steps.

  1. Open a terminal window and change directory to /etc/systemd/system where the Cloud HSM service unit file is located.
     cd /etc/systemd/system

     Example: azure-cloud-hsm.service



2. Start and Enable the Cloud HSM Client service

   sudo systemctl start azure-cloud-hsm.service
   sudo systemctl enable azure-cloud-hsm.service

3. Check status of the Cloud HSM Client service

   sudo systemctl status azure-cloud-hsm.service

```
chsmVMAdmin@myAdminVM:/etc/systemd/system$ sudo systemctl daemon-reload
chsmVMAdmin@myAdminVM:/etc/systemd/system$ sudo systemctl start azure-cloud-hsm.service
chsmVMAdmin@myAdminVM:/etc/systemd/system$ sudo systemctl enable azure-cloud-hsm.service
Created symlink /etc/systemd/system/multi-user.target.wants/azure-cloud-hsm.service → /e
tc/systemd/system/azure-cloud-hsm.service.
chsmVMAdmin@myAdminVM:/etc/systemd/system$ sudo systemctl status azure-cloud-hsm.service
● azure-cloud-hsm.service - Azure Cloud HSM Client Service
     Loaded: loaded (/etc/systemd/system/azure-cloud-hsm.service; enabled; vendor prese>
     Active: active (running) since Tue 2024-08-27 19:08:11 UTC; 918ms ago
   Main PID: 11230 (azcloudhsm_clie)
      Tasks: 3 (limit: 4625)
     Memory: 892.0K
     CGroup: /system.slice/azure-cloud-hsm.service
             └─11230 /opt/azurecloudhsm/bin/azcloudhsm_client /opt/azurecloudhsm/bin/az>

Aug 27 19:08:11 myAdminVM systemd[1]: Started Azure Cloud HSM Client Service.
```

4. Reload the Systemd configuration

```
sudo systemctl daemon-reload
sudo systemctl list-units --type=service --state=running
```

```
chsmVMAdmin@myAdminVM:/etc/systemd/system$ sudo systemctl list-units --type=service --state=running
  UNIT                      LOAD   ACTIVE SUB     DESCRIPTION
  accounts-daemon.service   loaded active running Accounts Service
  atd.service               loaded active running Deferred execution scheduler
  azure-cloud-hsm.service   loaded active running Azure Cloud HSM Client Service
  chrony.service            loaded active running chrony, an NTP client/server
```

Linux (Manual):

Start the client daemon if it is not running.

```
cd /opt/azurecloudhsm/bin
sudo ./azcloudhsm_client azcloudhsm_resource.cfg
```

You may also choose as an option with Linux to run the client daemon in the background using the following command.

```
sudo ./azcloudhsm_client azcloudhsm_resource.cfg > /dev/null 2>&1 &
```

- **Why am I getting error message invalid engine "azcloudhsm_openssl" could not load the shared library?**
  The error message suggests that the environment variable LD_LIBRARY_PATH has not been configured, and this configuration is necessary. To resolve this, you should set the LD_LIBRARY_PATH environment variable to the directory where the Azure Cloud HSM SDK is located.

- You will need to update your environment variables to reflect the correct path from the example below.
  export LD_LIBRARY_PATH=/opt/azurecloudhsm/lib64/:$LD_LIBRARY_PATH

- **Why am I getting error message environment variable azcloudhsm_openssl_conf is not set?**
  The error message signals that the prerequisite environment variable azcloudhsm_openssl_conf has not been configured. To resolve this issue, you should set the azcloudhsm_openssl_conf environment variable to the file path location of your azcloudhsm_openssl_dynamic.conf file within the Azure Cloud HSM SDK directory.
  - You will need to update your environment variables to reflect the correct path from the example below.
    export azcloudhsm_openssl_conf=/opt/azurecloudhsm/bin/azcloudhsm_openssl_dynamic.conf

- **Why am I getting error message environment variable azcloudhsm_password is not set?**
  The provided error message suggests that the prerequisite environment variable azcloudhsm_password has not been configured. To resolve this issue, it is necessary to set the azcloudhsm_password environment variable to the value of your cryptouser:password.
  - You will need to update your environment variables to reflect the correct crypto user and password from the example below.
    export azcloudhsm_password="cu1:user1234"

- **Why and I getting error message Key/Token not found or Invalid key-handle/token when attempting to Import Key?**
  The provided error message suggests that the specified wrapping key handle (i.e KEK Handle ID) is incorrect. The parameters '-f' specifies the filename containing the key to import, while '-w' specifies the wrapping key handle. During the creation and initialization of your Azure Cloud HSM you created a user generated KEK. The '-w' value should be the key handle id of your user generated KEK id.
  azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 importPrivateKey -l importTestKey -f key.pem -w 262150

- **Why am I getting error message azcloudhsm_application.cfg is not present?**
  To ensure the successful execution of your OpenSSL application, two conditions must be met. Firstly, the azcloudhsm_application.cfg file must be present in the same directory as your OpenSSL application. Secondly, the azcloudhsm_client should be running. If the azcloudhsm_client is not active, you will encounter a failed socket connection. Likewise, the absence of the azcloudhsm_application.cfg file will result in an error message. Depending on the execution location of your OpenSSL application, you may need to copy the azcloudhsm_application.cfg file from the Azure Cloud HSM SDK directory to the directory where you are running the OpenSSL application.

- **Why am I getting error message /configure: error: C compiler cc not found when trying to configure the Nginx sources?**
  The error message suggests that when you are trying to run configure command that you are missing C compiler from your system. Depending on your installation after running the following commands you may also need to install 'openssl libssl-dev' and 'zlib1g zlib1g-dev' which is a dependency.

- **For Red Hat based (using yum):**

```
sudo yum update
sudo yum groupinstall “Development Tools”
```

- **For Red Hat based systems (using dnf):**

```
sudo dnf update
sudo dnf install “Development Tools”
```

- **For Ubuntu based systems (using apt):**

```
sudo apt update
sudo apt install build-essential
```

- **Why am I getting error message /configure: error: SSL modules require the OpenSSL library?**
  The error message indicates that the OpenSSL library is required to enable SSL modules for Nginx. To resolve this issue, you can run the following command and then try to run configure again.

```
sudo apt-get install libssl-dev
```

- **Why am I getting error message /configure: error: the HTTP gzip module requires the zlib library?**
  The error message indicates that the zlib library is required for the HTTP gzip module in Nginx. To resolve this issue, you can run the following command and then try to run configure again.

```
sudo apt-get install zlib1g-dev
```

- **Why am I getting error message HSM Error: RET_USER_LOGIN_FAILURE?**
  The error message indicates that the specified user does not exist on one or more nodes where you're attempting a cryptographic operation. Although Azure Cloud HSM operates with a cluster of three nodes, the operation was directed to a node where the login failed. In such cases, a critical_err_info_* file is generated within the /opt/azurecloudhsm SDK directory.

```
cat critical_err_info_1710192371931969_1
    {
     "Command":"Login",
     "Cluster Status":[
      {
       "Node":"1",
       "Status":"HSM Return: SUCCESS"
```

```
  },
  {
    "Node":"2",
    "Status":"HSM Error: This user doesn't exist"
  },
  {
    "Node":"3",
    "Status":"HSM Error: This user doesn't exist"
  }
]
```

To mitigate this error, you need to create a 'user' with the Crypto User role that is replicated across all three nodes of your Azure Cloud HSM. This example shows starting azcloudhsm_mgmt_util and administrator logging on as CO, then proceeds to create 'cu1' user with crypto user role.

```
sudo ./azcloudhsm_mgmt_util ./azcloudhsm_resource.cfg
loginHSM CO admin adminpassword
createUser CU cu1 user1234
logoutHSM
loginHSM CU cu1 user1234
```

- **How do I display the contents of the CRT, CSR, and Key files I created?**
  - Display the contents of the CRT, CSR, or Key file you created in your terminal.
    ```
    cat filename.key
    ```
  - Display detailed information about a X.509 certificate contained in the PEM file. If your PEM file contains a private key or other types of data, you might need to adjust the command accordingly.
    ```
    openssl x509 -in filename.pem -text -noout
    ```
  - Display detailed information about a CSR, including the subject, public key, and any attributes included in the request.
    ```
    openssl req -in filename.csr -text -noout
    ```
  - Display detailed information about an RSA private key, including its modulus, public exponent, and other parameters.
    ```
    openssl rsa -in filename.key -text -noout
    ```

- Display detailed information about an EC private key, including the curve parameters, private key value, and other relevant information.
  openssl ec -in filename.key -text -noout

- **How do I generate an ED25519 key pair in Azure Cloud HSM for signing?**
  ED25519 keys are typically used for self-signed certificates or in certificate signing processes that directly use the private key. You can generate an ED25519 key pair using azcloudhsm_util.

  Important Note: Microsoft does not recommend using openssl genpkey or openssl ecparam. Using openssl genpkey or openssl ecparam with the -engine azcloudhsm_openssl to generate a ED25519 or ECCKeyPair will not produce an HSM key; instead, it will generate a software key. Customers needing ED25519 and other EC Key types must utilize azcloudhsm_util to create the key within the HSM.

  ./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 genECCKeyPair -i 28 -l labelED25519Test

- **Does getCaviumPrivKey extract the private key out of the HSM?**
  No. The private key is not being extracted from the HSM. azcloudhsm_util uses a private key handle ID as input. The getCaviumPrivKey command encodes the handle into a fake PEM format and outputs it to a file that can be used with the Azure Cloud HSM OpenSSL engine.

- **Can I use azcloudhsm_util to generate RSA and EC keys before using Azure Cloud HSM OpenSSL engine to generate CSR?**
  Yes. The following azcloudhsm_util commands can be run to create an RSA or EC key and then extract the private key to a fake pem format. Replace {PRIVATE_KEY_HANDLE} with the private key handle of the RSA or EC key you just created above. The private key meta file in PEM format does not contain any sensitive private key materials. It is just meta data that identifies the private key, and this meta file can only be understood by the Azure Cloud HSM OpenSSL engine.

  RSA Key:

  ```
  ./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 genRSAKeyPair -m 2048 -e 65537 -l labelRSATest
  ./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 getCaviumPrivKey -k {PRIVATE_KEY_HANDLE} -out
  web_server_fake_PEM.key
  openssl req -new -key web_server_fake_PEM.key -out web_server.csr -engine azcloudhsm_openssl
  openssl x509 -req -days 365 -in web_server.csr -signkey web_server_fake_PEM.key -out web_server.crt -engine azcloudhsm_openssl
  ```

  EC Key:

```
./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 genECCKeyPair -i 2 -l labelECTest
./azcloudhsm_util singlecmd loginHSM -u CU -p user1234 -s cu1 getCaviumPrivKey -k {PRIVATE_KEY_HANDLE} -out
web_server_fake_PEM.key
openssl req -new -key web_server_fake_PEM.key -out web_server.csr -engine azcloudhsm_openssl
openssl x509 -req -days 365 -in web_server.csr -signkey web_server_fake_PEM.key -out web_server.crt -engine azcloudhsm_openssl
```