Integrating OpenSSL with Azure Cloud HSM

Table of Contents

Summary

To use OpenSSL with Azure Cloud HSM for encryption, decryption, signing, and signature verification as an example, the Azure Cloud HSM Client SDK must be installed, and OpenSSL must be installed and configured on the same host that will be performing signing operations.

Important Note: Azure Cloud HSM OpenSSL Engine does not support Windows. Azure Cloud HSM OpenSSL Engine supports Linux only.

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, Ubuntu 24.04, RHEL 7, RHEL 8, RHEL 9, 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_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_openssl_conf=/opt/azurecloudhsm/bin/azcloudhsm_openssl_dynamic.conf
```

```
export LD_LIBRARY_PATH=/opt/azurecloudhsm/lib64/:$LD_LIBRARY_PATH
```

a. **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
```

```
Partition_Name=PARTITION_1
Username=cu1
Password=user1234
```

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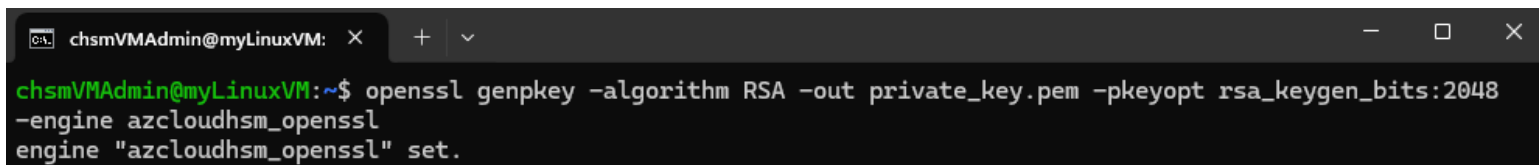   ```
   sudo ./azcloudhsm_client azcloudhsm_resource.cfg > /dev/null 2>&1 &
   ```

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
   ```

```
chsmVMAdmin@myLinuxVM:~$ openssl req -new -x509 -key private_key.pem -out certificate.crt -days 365 -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 []:
```

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


```
chsmVMAdmin@myLinuxVM:~$ echo "Azure Cloud HSM Plain Text File" >> plaintext.txt
chsmVMAdmin@myLinuxVM:~$ openssl rsautl -encrypt -pubin -inkey public_key.pem -in plaintext.txt -out enc_pl
aintext.txt -engine azcloudhsm_openssl
engine "azcloudhsm_openssl" set.
```

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
```


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.

- **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.

- o 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.
  - o 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 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 did I get error message 'signature did not match' when using OpenSSL or azcloudhsm_util to perform Sign/Verify with ECDSA?**
  By default, ECDSA in the Azure Cloud HSM OpenSSL Engine configuration is disabled. To enable it, you must update your azcloudhsm_openssl_dynamic.conf file. The configuration for ECDSA_Sign and ECDSA_Verify is set to 0 (disabled) by default. You'll need to modify these values to 1 for both Sign and Verify in the conf file to enable ECDSA functionality.

  ```
  ...
  ECDSA_Sign=1
  ECDSA_Verify=1
  ...
  ```

- **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.

OpenSSL Specifications

The Azure Cloud HSM OpenSSL engine only supports RSA key types. It does not support EC key generation. If you require both OpenSSL functionality and the generation of EC key types, you must utilize azcloudhsm_util to create the ECC Key Pair. azcloudhsm_util uses the private key handle ID generated 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.

> *Important Note: Using openssl ecparam with the -engine azcloudhsm_openssl to generate an ECCKeyPair will not produce an HSM key; instead, it will generate a software key. Customers needing both OpenSSL functionality and 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 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
```

Supported Key Types

The Azure Cloud HSM OpenSSL engine supports the following key types.

Key Type	Description
RSA	RSA key generation for 2048, 3072, and 4096-bit keys. RSA sign/verify. Verification is offloaded to OpenSSL software.
EC	ECDSA sign/verify for P-256, P-384, and secp256k1 key types. To ensure interoperability with the OpenSSL engine, customers should configure the azcloudhsm_openssl_dynamic.conf to support ECDSA and utilize azcloudhsm_util for generating the specified EC key types.

Common OpenSSL Use Case Examples

Using the OpenSSL command-line tool, you can securely sign files, documents and code using your private key and a digital certificate. While file signing, document signing, and code signing all involve the use of digital signatures and cryptographic techniques, they have distinct purposes and applications. File signing ensures the integrity and authenticity of various types of files. Document signing focuses on legal and business documents. Code signing is specific to software code and executables, emphasizing trust and security in the software supply chain. The OpenSSL command line options may be different depending on your scenario.

Important Note: For signing documents (such as text files, PDFs, or any other file), you can use the openssl dgst command to create a digital signature. The file/document signing process is typically used for non-email content and may require different algorithms or formats depending on your needs.

Important Note: For OpenSSL's "dgst" family of commands, the "-engine" option must be placed at the start of the command. In addition, the "-engine_impl" option should be provided to involve the Azure Cloud HSM OpenSSL engine in the digest implementation.

Important Note: For secure digital signature verification, you should use the signer's public key for verification rather than the signer's certificate directly. The use of the public key is the standard way to verify digital signatures.
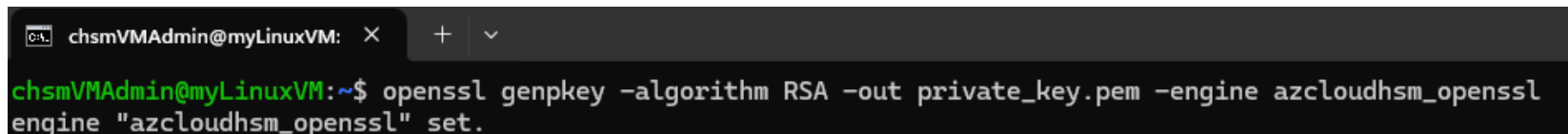
File Signing

File signing is a process of attaching a cryptographic signature to a file (i.e executable program, software update, script, configuration file) to verify its integrity and authenticity. File signing is commonly used in software distribution and security to ensure that files have not been tampered with during transit or storage. When you download software from a trusted source and it includes a digital signature, you can verify that the file has not been altered since it was signed. File signing often employs digital signatures generated with asymmetric cryptography (e.g., RSA, ECC) and is commonly associated with code signing certificates for software publishers.

The following steps explain how to sign a file with OpenSSL when working with Azure Cloud HSM.

1. **Generate a Private Key.** This command generates a new RSA private key and saves it to the file private_key.pem

   ```
   cd ~
   openssl genpkey -algorithm RSA -out private_key.pem -engine azcloudhsm_openssl
   ```



```
chsmVMAdmin@myLinuxVM:~$ openssl genpkey -algorithm RSA -out private_key.pem -engine azcloudhsm_openssl
engine "azcloudhsm_openssl" set.
```

2. **Create a Certificate Signing Request (CSR)**

   ```
   openssl req -new -key private_key.pem -out cert.csr -engine azcloudhsm_openssl
   ```

```
chsmVMAdmin@myLinuxVM:~$ openssl req -new -key private_key.pem -out cert.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 []:
```

3. **Create a Self-Signed Certificate**

   openssl x509 -req -days 365 -in cert.csr -signkey private_key.pem -out certificate.pem -engine azcloudhsm_openssl


```
chsmVMAdmin@myLinuxVM:~$ openssl x509 -req -days 365 -in cert.csr -signkey private_key.pem -out certificate.pem -engine azcloudhsm_
openssl
engine "azcloudhsm_openssl" set.
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
Getting Private key
```
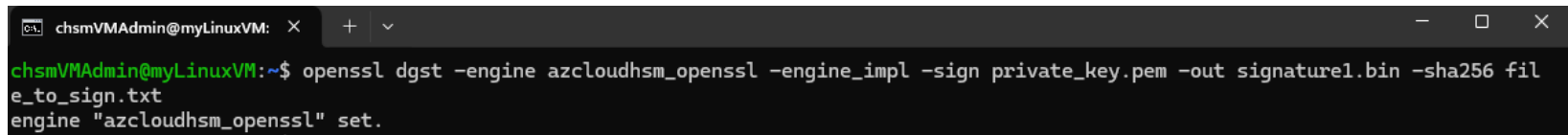
4. **Prepare the document.** Make sure you have the file you want to sign ready in a suitable format. Ensure that the file is in a format that supports digital signatures.

- ***Create a sample plain text file for signing.*** *This bullet step is included for testing purposes only. In the next steps below we're going to use our private key to sign this file_to_sign.txt. You may choose to sign any file following these steps that is in a suitable format.*

  ```
  echo "Azure Cloud HSM Plain Text File" > file_to_sign.txt
  ```

5. **Sign the file using the private key.** Use the opennssl dgst command to create a digital signature of the file. You will need your private key for signing. Replace file_to_sign.txt with the file you want to sign.
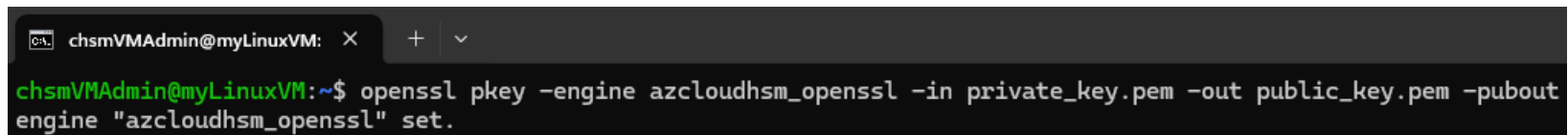
   ```
   openssl dgst -engine azcloudhsm_openssl -engine_impl -sign private_key.pem -out signature1.bin -sha256 file_to_sign.txt
   ```

   ```
   chsmVMAdmin@myLinuxVM:~$ openssl dgst -engine azcloudhsm_openssl -engine_impl -sign private_key.pem -out signature1.bin -sha256 fil
   e_to_sign.txt
   engine "azcloudhsm_openssl" set.
   ```

6. **Extract Public Key.** This command extracts the public key from private_key.pem and saves it to public_key.pem.
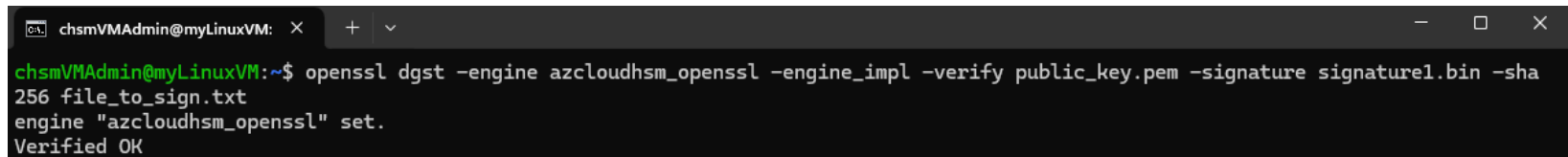
   ```
   openssl pkey -engine azcloudhsm_openssl -in private_key.pem -out public_key.pem -pubout
   ```

   ```
   chsmVMAdmin@myLinuxVM:~$ openssl pkey -engine azcloudhsm_openssl -in private_key.pem -out public_key.pem -pubout
   engine "azcloudhsm_openssl" set.
   ```

7. **Verify the signature.** Replace file_to_sign.txt with the file that you signed. When successful this will return "Verified OK".

   ```
   openssl dgst -engine azcloudhsm_openssl -engine_impl -verify public_key.pem -signature signature1.bin -sha256 file_to_sign.txt
   ```

   ```
   chsmVMAdmin@myLinuxVM:~$ openssl dgst -engine azcloudhsm_openssl -engine_impl -verify public_key.pem -signature signature1.bin -sha
   256 file_to_sign.txt
   engine "azcloudhsm_openssl" set.
   Verified OK
   ```
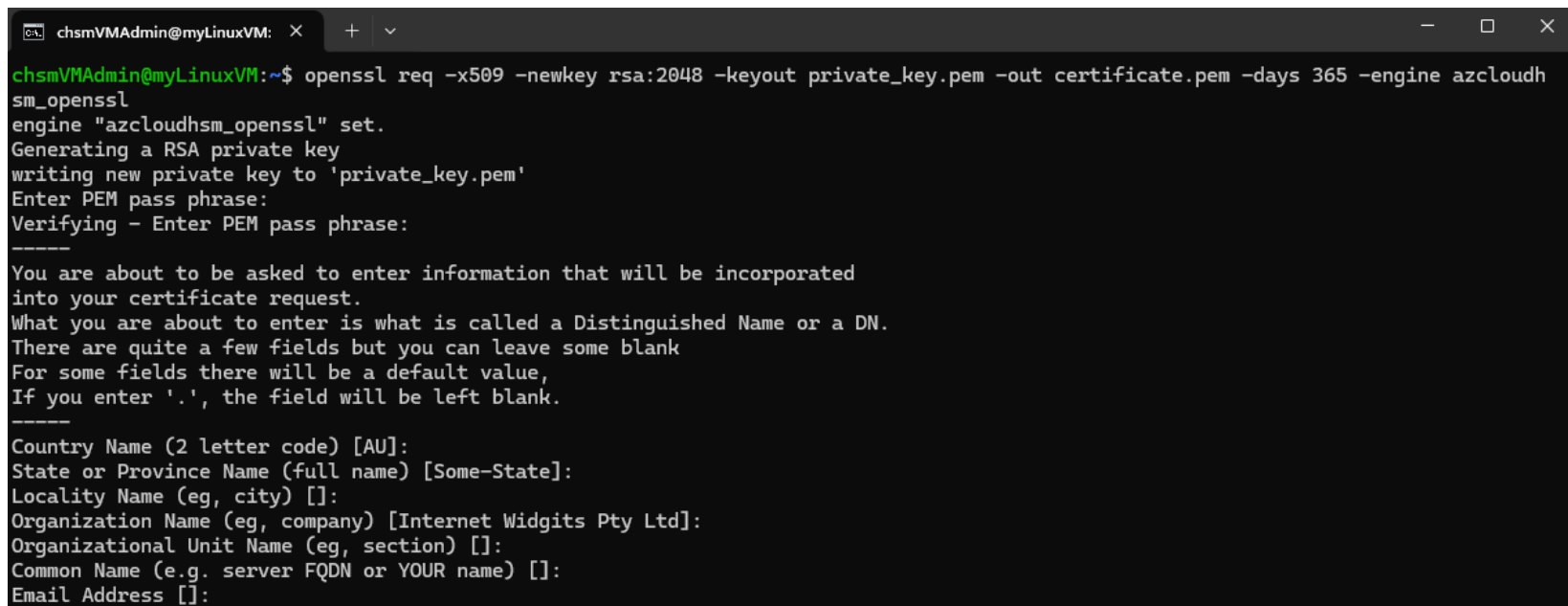
Document Signing

Document signing is the process of applying a digital signature to a document (e.g., PDF, Word document, contract) to confirm its authenticity, integrity, and the identity of the signer. Document signing is frequently used in situations where it is essential to prove that a document has not been altered since it was signed and that it was signed by a specific person or entity. It is common in legal contracts, digital agreements, and

other situations where electronic documents need to be legally binding. Document signing also uses digital signatures, typically generated with asymmetric cryptography, but the focus is on the content of the document and ensuring it remains unchanged.

The following steps explain how to sign a document with OpenSSL when working with Azure Cloud HSM.

1. **Generate Private Key and Certificate.** If you don't already have a private key and certificate, you'll need to generate one. Using the following OpenSSL commands will generate a self-signed certificate (certificate.pem) along with a private key (private_key.pem).

   ```
   cd ~
   openssl req -x509 -newkey rsa:2048 -keyout private_key.pem -out certificate.pem -days 365 -engine azcloudhsm_openssl
   ```
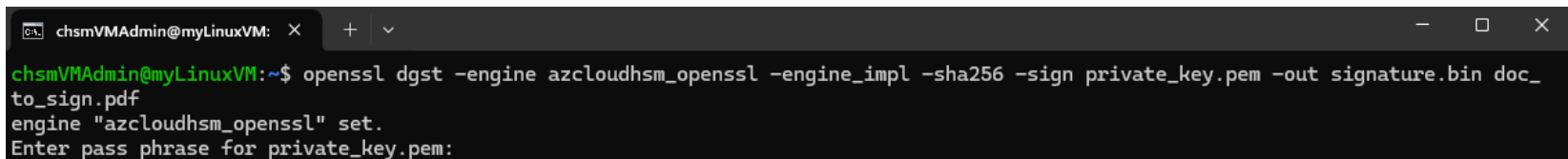

2. **Prepare the document.** Make sure you have the document you want to sign ready in a suitable format. Ensure that the file is in a format that supports digital signatures.
   - *Create a sample PDF document for signing. This bullet step is included for testing purposes only. In the next steps below we're going to use our private key to sign this doc_to_sign.pdf. This example makes use of enscript and ghostscript to create the PDF test file. You may choose to sign any document following these steps that is in a suitable format.*

```
sudo apt install enscript
sudo apt install ghostscript
echo "Azure Cloud HSM PDF File" | enscript -o - | ps2pdf - doc_to_sign.pdf
```

3. **Sign the Document.** Use the dgst command to sign the document using your private key. You'll also need to specify the hash algorithm you want to use. In our example, we will sign with SHA-256. You can replace it with other supported algorithms like sha512, etc., depending on your requirements.
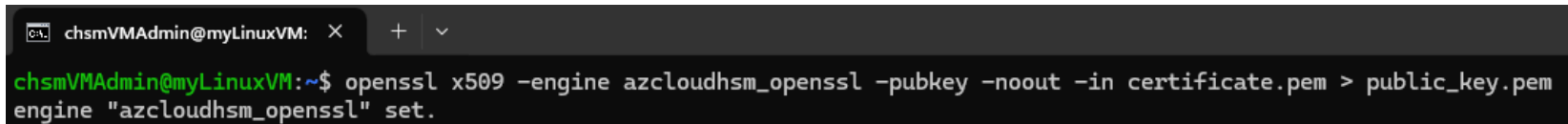   ```
   openssl dgst -engine azcloudhsm_openssl -engine_impl -sha256 -sign private_key.pem -out signature.bin doc_to_sign.pdf
   ```

```
chsmVMAdmin@myLinuxVM:  ×     +   ∨                                                              –    □    ×

chsmVMAdmin@myLinuxVM:~$ openssl dgst —engine azcloudhsm_openssl —engine_impl —sha256 —sign private_key.pem —out signature.bin doc_
to_sign.pdf
engine "azcloudhsm_openssl" set.
Enter pass phrase for private_key.pem:
```

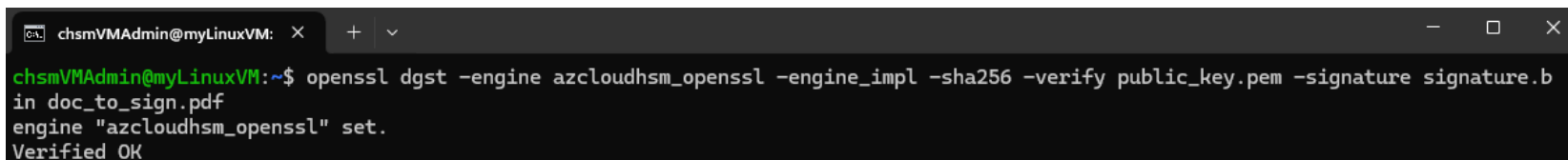5. **Extract Public Key.** This command extracts the public key from certificate.pem and saves it to public_key.pem.
   ```
   openssl x509 -engine azcloudhsm_openssl -pubkey -noout -in certificate.pem > public_key.pem
   ```

```
chsmVMAdmin@myLinuxVM:  ×     +   ∨

chsmVMAdmin@myLinuxVM:~$ openssl x509 —engine azcloudhsm_openssl —pubkey —noout —in certificate.pem > public_key.pem
engine "azcloudhsm_openssl" set.
```

6. **Verify the Signature.** You can verify the signature on the document using the openssl dgst command.
   ```
   openssl dgst -engine azcloudhsm_openssl -engine_impl -sha256 -verify public_key.pem -signature signature.bin doc_to_sign.pdf
   ```

```
chsmVMAdmin@myLinuxVM:  ×     +   ∨                                                              –    □    ×

chsmVMAdmin@myLinuxVM:~$ openssl dgst —engine azcloudhsm_openssl —engine_impl —sha256 —verify public_key.pem —signature signature.b
in doc_to_sign.pdf
engine "azcloudhsm_openssl" set.
Verified OK
```

Code Signing

Code signing is a specialized form of file signing that is used specifically to sign software code, scripts, and executables. It verifies the authenticity of the code's author and ensures that the code has not been modified since it was signed. Code signing is crucial in software security. It allows software developers and publishers to prove their identity, indicating that the code is from a trusted source. Users can then verify the signature

to confirm that the software has not been tampered with or altered by malicious actors. Code signing is commonly used for signed scripts (e.g., PowerShell scripts), Java applets, mobile apps, and software installers.

OpenSSL itself doesn't provide direct support for code signing because code signing typically involves platform-specific tools and certificates issued by trusted Certificate Authorities (CAs). Code signing is commonly used to ensure the authenticity and integrity of software code and executables. However, if you have specific requirements to use OpenSSL in your code signing process, you can incorporate it as part of a larger workflow.

The following steps explain how to code sign with OpenSSL when working with Azure Cloud HSM.

1. **Create a sample DEB package for code signing.** This bullet step is included for testing purposes only. First steps below you're going to create a deb package. You may choose to sign any applicable code file that is in a suitable format. In our examples below the package we created was mypackage.deb.

2. **Generate a Certificate Signing Request (CSR).** Before you can sign your code, you need a digital certificate. You can generate a CSR using the following OpenSSL command. This command generates a private key (private_key.pem) and a CSR (mycsr.csr) that you'll use to request a certificate from a Certificate Authority (CA).

   ```
   cd ~
   openssl req -newkey rsa:2048 -keyout private_key.pem -out mycsr.csr -engine azcloudhsm_openssl
   ```

```
chsmVMAdmin@myLinuxVM:~$ openssl req -newkey rsa:2048 -keyout private_key.pem -out mycsr.csr -engine azcloudhsm_openssl
engine "azcloudhsm_openssl" set.
Generating a RSA private key
writing new private key to 'private_key.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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 []:
```
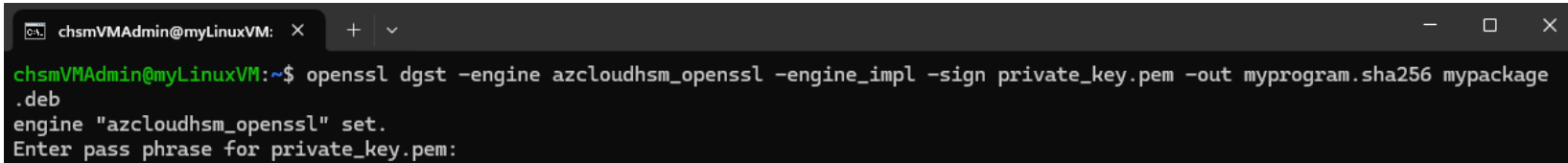
3. **Submit the CSR to a Certificate Authority (CA).** You can either self-sign the certificate or get one from a trusted CA. If you're using a CA, you'll typically follow their instructions to obtain a code signing certificate. If you're self-signing, you can use the following command below.

   openssl x509 -req -in mycsr.csr -signkey private_key.pem -out mycert.crt -engine azcloudhsm_openssl


```
chsmVMAdmin@myLinuxVM:~$ openssl x509 -req -in mycsr.csr -signkey private_key.pem -out mycert.crt -engine azcloudhsm_openssl
engine "azcloudhsm_openssl" set.
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
Getting Private key
Enter pass phrase for private_key.pem:
```

4. **Sign Your Code.** Once you have a certificate (self-signed or from a CA), you can use it to sign your code.
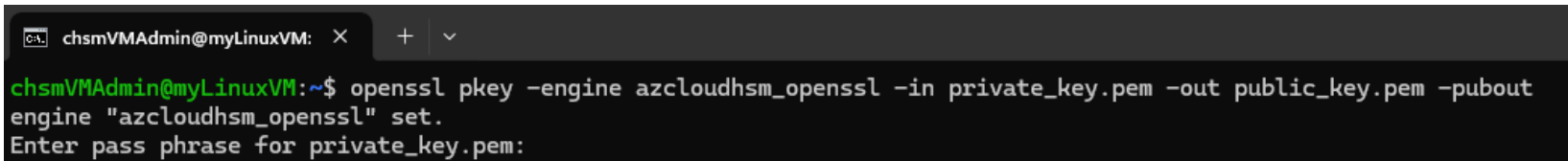
```
openssl dgst -engine azcloudhsm_openssl -engine_impl -sign private_key.pem -out myprogram.sha256 mypackage.deb
```

```
chsmVMAdmin@myLinuxVM:    ✕    +  ✓                                                      —   ☐   ✕

chsmVMAdmin@myLinuxVM:~$ openssl dgst -engine azcloudhsm_openssl -engine_impl -sign private_key.pem -out myprogram.sha256 mypackage
.deb
engine "azcloudhsm_openssl" set.
Enter pass phrase for private_key.pem:
```

5. **Extract Public Key.** This command extracts the public key from private_key.pem and saves it to public_key.pem.
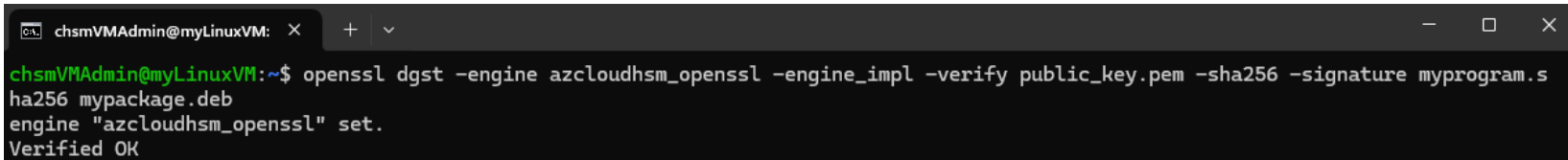```
openssl pkey -engine azcloudhsm_openssl -in private_key.pem -out public_key.pem -pubout
```

```
chsmVMAdmin@myLinuxVM:    ✕    +  ✓

chsmVMAdmin@myLinuxVM:~$ openssl pkey -engine azcloudhsm_openssl -in private_key.pem -out public_key.pem -pubout
engine "azcloudhsm_openssl" set.
Enter pass phrase for private_key.pem:
```

6. **Verify the Signature.** You can verify the signature on the document using the openssl dgst command.
```
openssl dgst -engine azcloudhsm_openssl -engine_impl -verify public_key.pem -sha256 -signature myprogram.sha256 mypackage.deb
```

```
chsmVMAdmin@myLinuxVM:    ✕    +  ✓                                                      —   ☐   ✕

chsmVMAdmin@myLinuxVM:~$ openssl dgst -engine azcloudhsm_openssl -engine_impl -verify public_key.pem -sha256 -signature myprogram.s
ha256 mypackage.deb
engine "azcloudhsm_openssl" set.
Verified OK
```

7. **Distribute Your Signed Code.** After signing your code, distribute it as needed. Users can verify the signature to ensure the code hasn't been tampered with by checking the digital signature against your certificate. Remember to keep your private key secure, as it's used to sign your code and should not be shared or compromised.

Email Message Signing

Email message signing with OpenSSL involves using digital signatures to verify the authenticity and integrity of an email message. It provides a way for the recipient to ensure that the email was sent by the claimed sender and that the message content hasn't been tampered with during transit.

Important Note: S/MIME signing typically involves certificates issued by a trusted certificate authority (CA) for secure communication. In a production environment, you would use a CA-issued certificate instead of a self-signed one for enhanced security and trust. For the example below we will be using a self-signed certificate.

Important Note: For S/MIME verification, the certificate used to sign the document must be created with the "digital signature" attribute in the key usage. The cert.cnf file must have an entry in it for "keyUsage", for example:

1. **Create a cert.cnf configuration file.** This configuration file is used by OpenSSL for generating X.509 certificates and requests. When you use OpenSSL to create a certificate signing request (CSR) or a self-signed certificate, you can provide a configuration file to specify the details of the certificate.

```
[ req ]
distinguished_name = req_distinguished_name
x509_extensions = v3_req
prompt = no
string_mask = utf8only
utf8 = yes

[ req_distinguished_name ]
countryName          = CN
stateOrProvinceName    = SN
localityName         = LN
organizationName      = ORG
organizationalUnitName = UN
commonName           = Name
emailAddress          = someone@somewhere.com

[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
```
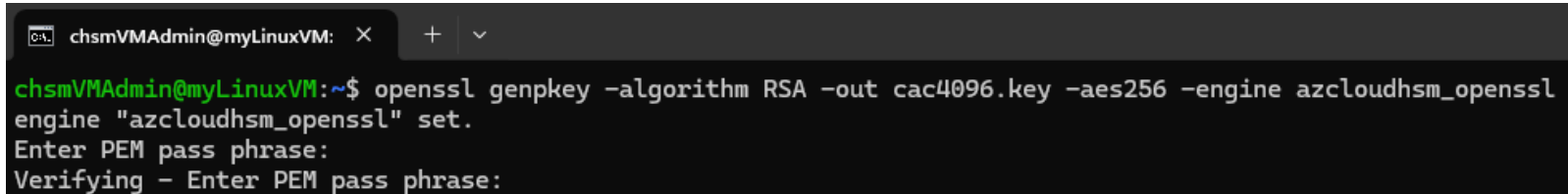
2. **Create a sample message for signing.**
```
echo "Azure Cloud HSM Email Message" > unsigned_message.txt
```

3. **Create a Private Key.** When you run this command, OpenSSL will prompt you to enter a passphrase for the private key. After running this command, you'll have a new RSA private key file (cac4096.key) in the specified location, encrypted with AES-256.
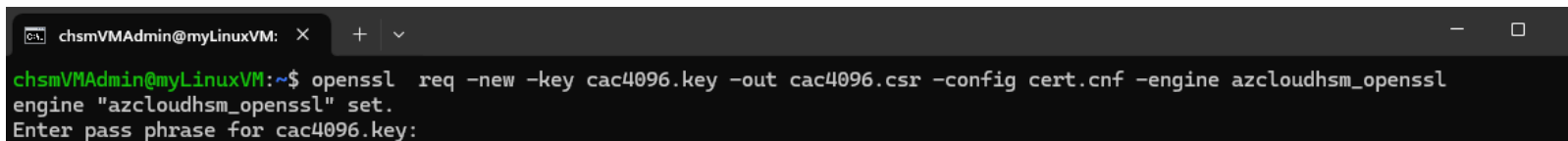
    openssl genpkey -algorithm RSA -out cac4096.key -aes256 -engine azcloudhsm_openssl

```
chsmVMAdmin@myLinuxVM:    ×    +    ∨
chsmVMAdmin@myLinuxVM:~$ openssl genpkey -algorithm RSA -out cac4096.key -aes256 -engine azcloudhsm_openssl
engine "azcloudhsm_openssl" set.
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

4. **Create a Certificate Signing Request.** This command will generate a new certificate signing request (CSR) based on an existing private key (cac4096.key) and a configuration file (cert.cnf).
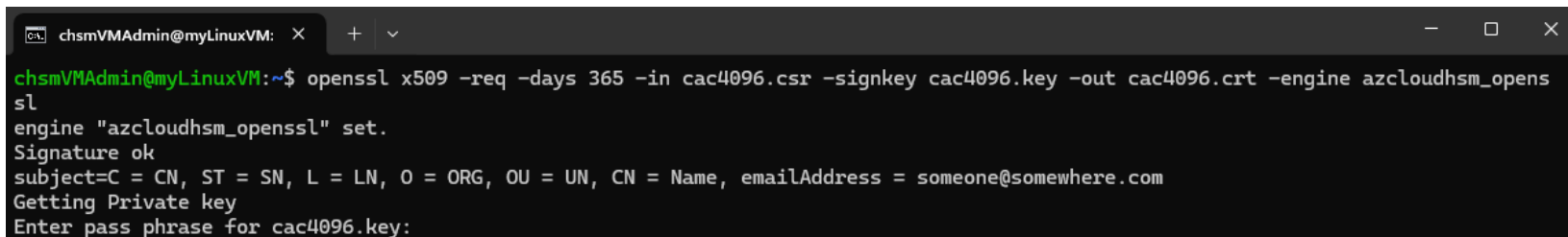
    openssl req -new -key cac4096.key -out cac4096.csr -config cert.cnf -engine azcloudhsm_openssl

```
chsmVMAdmin@myLinuxVM:    ×    +    ∨                                              —    □    ×
chsmVMAdmin@myLinuxVM:~$ openssl  req -new -key cac4096.key -out cac4096.csr -config cert.cnf -engine azcloudhsm_openssl
engine "azcloudhsm_openssl" set.
Enter pass phrase for cac4096.key:
```

5. **Generate a signed X.509 certificate.** This command is used for signing a CSR with a private key, resulting in the generation of a self-signed X.509 certificate.

    openssl x509 -req -days 365 -in cac4096.csr -signkey cac4096.key -out cac4096.crt -engine azcloudhsm_openssl
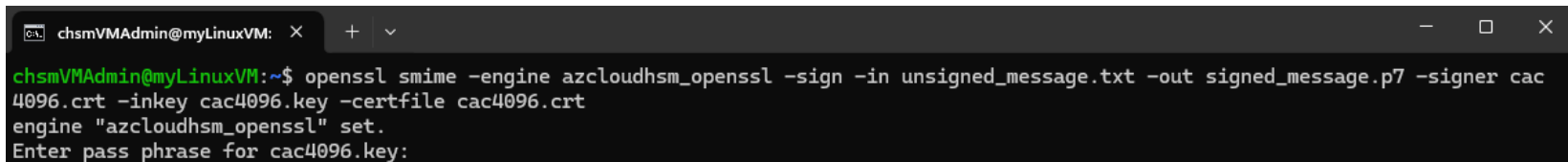
```
chsmVMAdmin@myLinuxVM:    ×    +    ∨                                              —    □    ×
chsmVMAdmin@myLinuxVM:~$ openssl x509 -req -days 365 -in cac4096.csr -signkey cac4096.key -out cac4096.crt -engine azcloudhsm_opens
sl
engine "azcloudhsm_openssl" set.
Signature ok
subject=C = CN, ST = SN, L = LN, O = ORG, OU = UN, CN = Name, emailAddress = someone@somewhere.com
Getting Private key
Enter pass phrase for cac4096.key:
```

6. **Verify signed X.509 certificate.** This command will display the textual representation of an X.509 certificate without printing the encoded version. This is a useful command for inspecting the details of a certificate, including its metadata and public key information.

```
openssl x509 -in cac4096.crt -text -noout
```

7. **Sign the sample message.** This command is used to sign a message (in this case, a text file) using S/MIME. After running this command, you will have a file (signed_message.p7) containing the signed message in PKCS#7 format. This format typically includes the signed content and the signer's certificate.
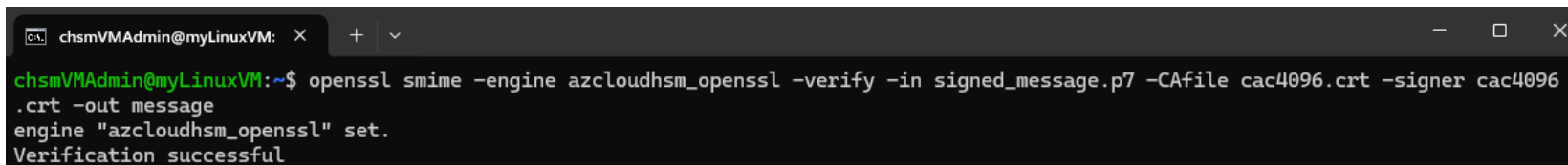
```
openssl smime -engine azcloudhsm_openssl -sign -in unsigned_message.txt -out signed_message.p7 -signer cac4096.crt -inkey cac4096.key -certfile cac4096.crt
```

```
chsmVMAdmin@myLinuxVM:  ×    +   ∨                                              —   □   ×

chsmVMAdmin@myLinuxVM:~$ openssl smime -engine azcloudhsm_openssl -sign -in unsigned_message.txt -out signed_message.p7 -signer cac
4096.crt -inkey cac4096.key -certfile cac4096.crt
engine "azcloudhsm_openssl" set.
Enter pass phrase for cac4096.key:
```

8. **Verify the signed sample message.** This command is used to verify a signed S/MIME message. After running this command, if the signature is valid and the signer's certificate is trusted, it will output the verified content to the specified output file.

```
openssl smime -engine azcloudhsm_openssl -verify -in signed_message.p7 -CAfile cac4096.crt -signer cac4096.crt -out message
```

```
chsmVMAdmin@myLinuxVM:  ×    +   ∨                                              —   □   ×

chsmVMAdmin@myLinuxVM:~$ openssl smime -engine azcloudhsm_openssl -verify -in signed_message.p7 -CAfile cac4096.crt -signer cac4096
.crt -out message
engine "azcloudhsm_openssl" set.
Verification successful
```