Microsoft Azure Cloud HSM Onboarding Guide

Table of Contents

Summary

Microsoft Azure Cloud HSM is a service that provides secure storage for cryptographic keys using hardware security modules that meet the FIPS 140-3 Level 3 security standard. It is a customer-managed single-tenant, highly available service that is compliant with industry standards.

Microsoft Azure Cloud HSM provides secure and customer managed HSM for storing cryptographic keys and performing cryptographic operations. It supports various applications, including PKCS#11, offload SSL/TLS processing, certificate authority private key protection, transparent data encryption, including document and code signing.

Microsoft Azure Cloud HSM provides high availability and redundancy by grouping multiple HSMs into an HSM cluster and automatically synchronizing across 3 HSM instances. The HSM cluster supports load balancing of cryptographic operations, and periodic HSM backups ensure secure and simple data recovery.

Prerequisites

We recommend customers to have their VNET, Private Endpoint, Cloud HSM, and Client VM deployed in the same designated region.

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

Important Note: All onboarding and deployment steps and examples below should be executed from Azure Portal within Cloud Shell

Supported Operating Systems

The Azure Cloud HSM SDK currently only supports the following Operating Systems.

- Windows (Server 2016, 2019, 2022, 2025)
- Linux (Ubuntu 20.04, Ubuntu 22.04, Ubuntu 24.04, RHEL 7, RHEL 8, RHEL9)
- CBL Mariner 2

Important Note: Any other operating system unlisted above is not supported by Azure Cloud HSM currently.

- *CentOS 7 is not supported! Red Hat no longer supports CentOS 7 as it reached end of life June 30th, 2024*
- *CentOS 8 is not supported! Red Hat no longer supports CentOS 8 as it reached end of life December 31st, 2021.*
- *Ubuntu 18.04 is not supported! Ubuntu no longer supports 18.04 as it reached end of life April 30th, 2023.*

Azure Cloud HSM Onboarding

STEP 1: Choosing the right Azure HSM solution

Azure offers multiple solutions for cryptographic key storage and management in the cloud: Azure Key Vault (standard and premium offerings), Azure Managed HSM, Azure Payment HSM and Azure Cloud HSM. It may be overwhelming for customers to decide which key management solution is correct for them.

Azure Cloud HSM is the most suitable for migration scenarios. This means if you are migrating on-premises applications to Azure, and you are already using HSMs. This provides a low-friction option to migrate to Azure with minimal changes to the application. If cryptographic operations are performed in the application's code running in Azure VM or Web App, Cloud HSM may be used. In general, shrink-wrapped software running in IaaS (infrastructure as a service) models that support HSMs as a key store can use Cloud HSM, such as:

- ADCS (Active Directory Certificate Services)
- SSL/TLS Offload for Nginx and Apache
- Tools/applications used for document signing.
- Code signing.
- Java applications that require JCE provider
- Microsoft SQL Server TDE (IaaS) through EKM.
- Oracle TDE

Deploy Azure Cloud HSM

Microsoft Azure Cloud HSM supports Azure CLI, Azure PowerShell, and ARM deployments for Cloud HSM resources.

Important Note: After creating a key, wait at least 24 hours to allow synchronization and backups to fully complete across your Azure Cloud HSM deployment. This ensures the key is durable, present on all HSM nodes, and included in backups, preventing issues such as missing or unsynchronized keys during restore or recovery operations, or when an application connects to a node that has not yet received the key.

STEP 2: Deploy Azure Cloud HSM

The following example will create a resource group and your first Cloud HSM instance. When using the example below, you will need to update the Subscription, Resource Group, and Location accordingly and HSM Name to a unique resource name. If you specify a resource name for your HSM that already exists in the given region your deployment will fail.

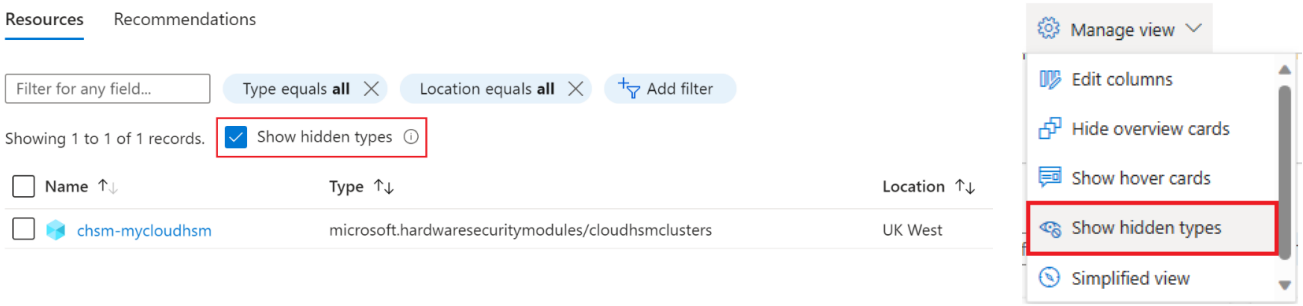
We recommend deploying your Azure Cloud HSM resource in a separate resource group, then your related Client VNET, and VM resources. In this onboarding guide we will use CHSM-SERVER-RG as an example.

Deploy Azure Cloud HSM

```
$server = @{
   Location = "<RegionName>"
   Sku = @{"family" = "B"; "Name" = "Standard_B1" }
   ResourceName = "<HSMName>"
   ResourceType = "microsoft.hardwaresecuritymodules/cloudHsmClusters"
   ResourceGroupName = "<ResourceGroupName>"
   Force = $true
}

#Create HSM Cluster Resource Group
New-AzResourceGroup -Name $server.ResourceGroupName -Location $server.Location -Force

#Create HSM Cluster
New-AzResource @server -AsJob -Verbose
```

STEP 3: Validate Cloud HSM resource was created.

Go to Azure Portal and open the Server Resource Group you just created and select “Show Hidden Types”. If you are not in classic view mode, then the option to select “Show Hidden Types” is found under manage view. You will see “microsoft.hardwaresecuritymodules/cloudhsmclusters” was created with your given HSM resource name in your designated region.



Important Note: Click on your Azure Cloud HSM resource. You will see provisioning state shows “provisioning”. The Azure Cloud HSM resource is not fully deployed until you see provisioning state changed to “Succeeded”. Do not proceed to Step 4 (Create Client VNET, Private DNS Zone, and Private Endpoint) until provisioning state shows “Succeeded” to ensure next steps are successful when applying private endpoint.

Create Client VNET & Private DNS Zone / Private Endpoint

STEP 4: Create VNET and Private DNS Zone

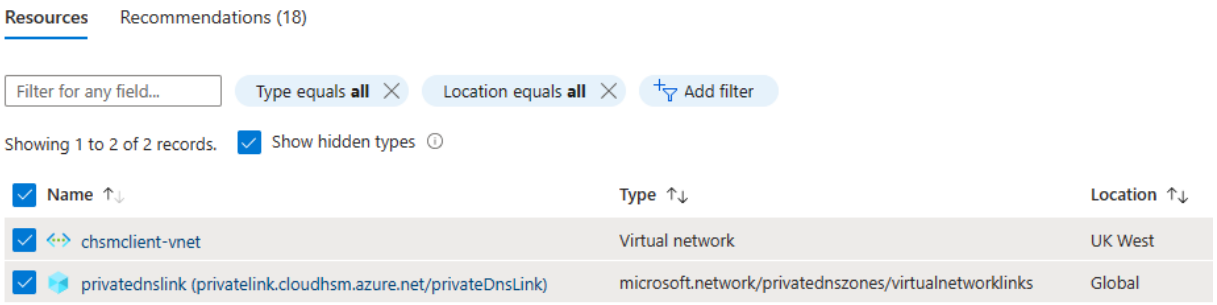
We recommend customers to have their Client VNET and VM deployed in the same designated region as their HSM resource. When using the example below, you will need to update the location and resource group accordingly. The following PowerShell example will create a resource group and private DNS zone. In this onboarding guide we will use CHSM-CLIENT-RG as Resource Group, and chsmclient-vnet as Virtual Network Name as an example.

```
$client = @{
    Location          = "<RegionName>"
    ResourceGroupName = "<ResourceGroupName>"
    ZoneName          = "privatelink.cloudhsm.azure.net"
```

```
    VirtualNetworkName = "<VirtualNetworkName>"
}

#Create HSM Client Resource Group
New-AzResourceGroup -Name $client.ResourceGroupName -Location $client.Location -Force

#Create VNET and Private DNS Zone
$zone = New-AzPrivateDnsZone -ResourceGroupName $client.ResourceGroupName -Name $client.ZoneName

$vnet = New-AzVirtualNetwork -Name $client.VirtualNetworkName -ResourceGroupName
$client.ResourceGroupName -Location $client.Location -AddressPrefix '10.0.0.0/16'

$subnet = Add-AzVirtualNetworkSubnetConfig -Name "default" -VirtualNetwork $vnet -AddressPrefix
"10.0.2.0/24" | Set-AzVirtualNetwork

New-AzPrivateDnsVirtualNetworkLink -ResourceGroupName $client.ResourceGroupName -ZoneName
$client.ZoneName -Name privateDnsLink -VirtualNetworkId $vnet.Id
```

STEP 5: Validate VNET and Private DNS Zone was created.

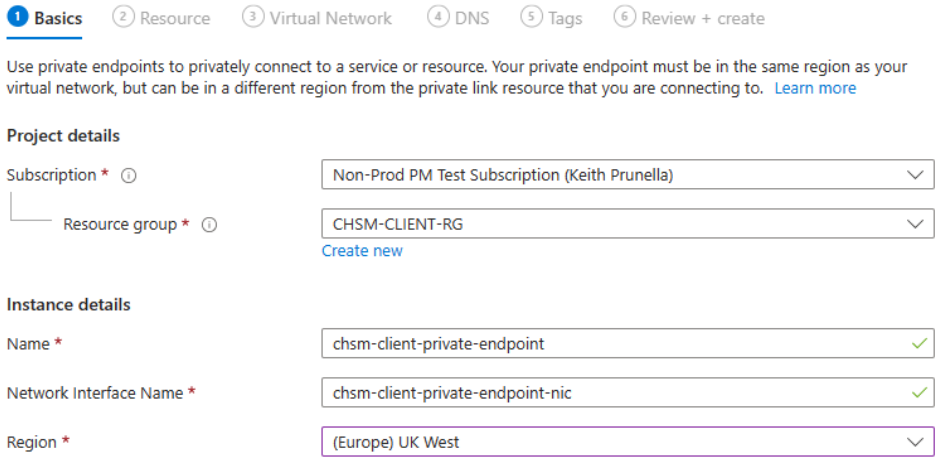
Go to Azure Portal and open the Client Resource Group you just created. You will see your VNET was created in your designated region and the Private DNS Zone as global. If you select your virtual network and select subnets within the left menu under settings, you will see the address prefix you specified at the time of creation.



STEP 6: Create Private Endpoint for Azure Cloud HSM

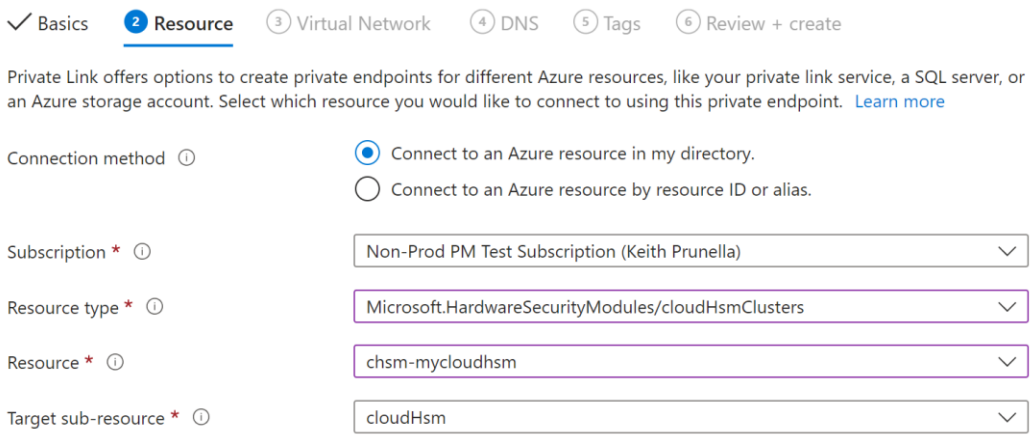
To create a Private Endpoint for your Azure Cloud HSM you will need to complete the following steps within the Azure Portal.

1. Select your Client Resource Group. In this example it is CHSM-CLIENT-RG
2. Click 'Create'
3. Search 'Private Endpoint'
4. Click 'Create' dropdown for private endpoint and select 'Private Endpoint' option.
5. **Basics**
    a. Protect Details
        i. Ensure you select your customer subscription under project details for Azure Cloud HSM.
        ii. For 'Resource Group' ensure to select your 'Client Resource Group'
    b. Instance Details
        i. Provide a name for your private endpoint. In this example we will use 'chsm-client-private-endpoint'
        ii. For region ensure to select your designated region.



6. **Resource**
    a. Leave 'Connection method' as default which is "Connect to an Azure resource in my directory."
    b. Ensure you select your customer subscription for Azure Cloud HSM.
    c. For 'Resource type' search for "Cloud" and select 'Microsoft.HardwareSecurityModules/cloudHsmClusters'
    d. For 'Resource' select your HSM name. In this example we will use '`chsm-mycloudhsm`'

e. ‘Target sub-resource’ should show as ‘cloudHSM’

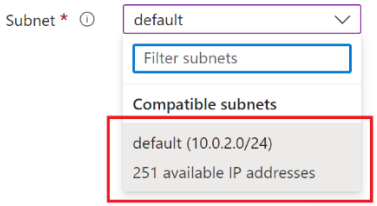


7. Virtual Network

a. **Networking**

i. For ‘Virtual network’ it will auto populate with your client VNET.

ii. Leave ‘Subnet’ as default. If you select ‘Subnet’ you will see default shows as the IP Address prefix specified during creation.



b. Private IP Configuration

i. Leave ‘Dynamically allocate IP Address’ as default.

c. Application Security Group

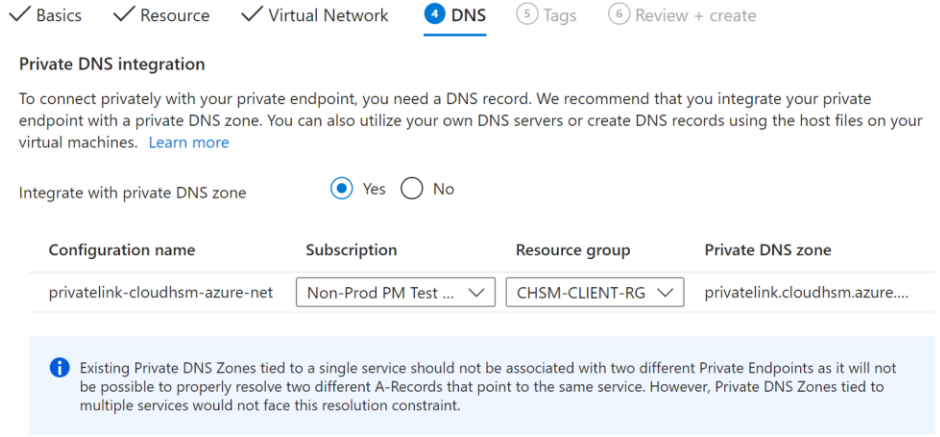
i. Leave Blank. Do Not Create. Not required.

8. DNS

a. ‘Integrate with private DNS zone’ should be set to “Yes”

b. You ‘Subscription’ and ‘Resource Group’ you should see auto populate.

i. If ‘Configuration Name’, ‘Subscription’, ‘Resource Group’, ‘Private DNZ Zone’ look correct then no action is required in this section of the wizard, you can click “Next”.



9. TAGS

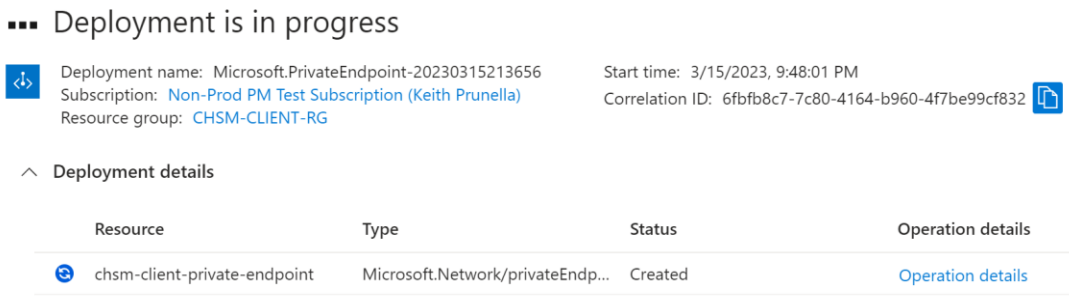
a. Skip ‘Tags’ and go to “Review + Create.”

10. REVIEW+CREATE

a. Review the metadata shown derived from the information you provided during the private endpoint wizard to ensure all information is correct.

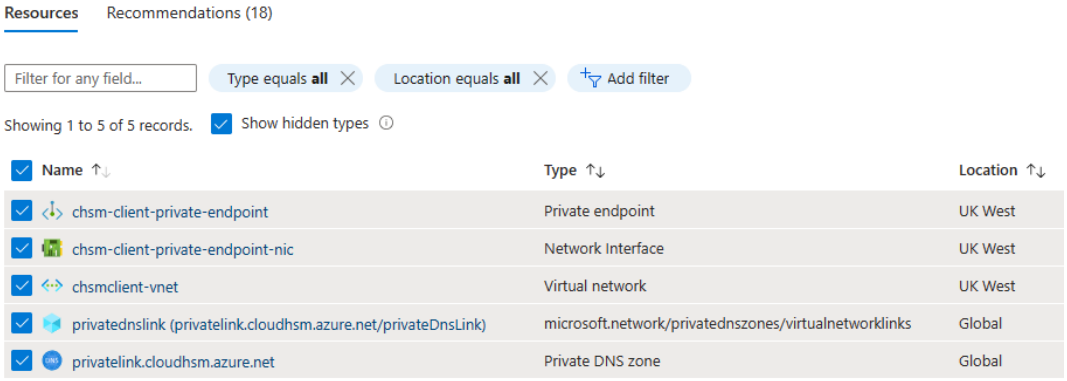
b. Click ‘Create’.

i. You will then be directed to a status page showing “Deployment is in progress”. It will change to “Your deployment is complete” once finished.

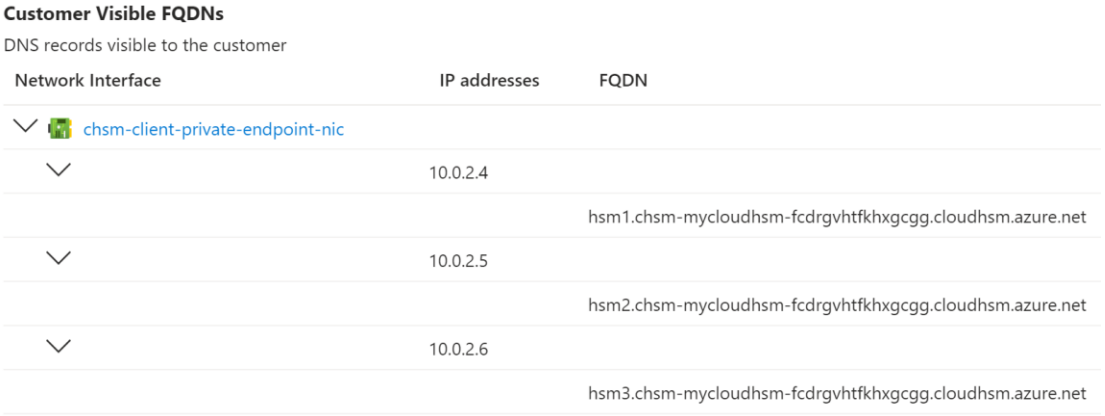


STEP 7: Validate Private Endpoint Created

Go to Azure Portal and open your Client Resource Group you created and select “Show Hidden Types”. You will see ‘privatednslink’ located as global and your private endpoint and network interface (chsm-client-private-endpoint-nic) you just created located in your designated region.



Selecting ‘Private Endpoint’ in this example it is ‘chsm-client-private-endpoint’ you will see the ‘chsm-client-private-endpoint-nic’ has 3 private IP addresses, each corresponds to 1 of the HSM partitions in the HSM cluster. The random generated suffix is added to be compliant with the unique FQDN requirement for Azure Resource Manager.



Create Admin Virtual Machine in Client VNET

STEP 8: Create an Admin VM in your Private VNET

You will need to create a Virtual Machine for your client VNET. This will serve as your Admin VM workstation. After provisioning the VM you will download the Azure Cloud HSM SDK from GitHub to this VM to configure your Cloud HSM. You can find details on how to deploy a Windows or Linux Virtual Machine using Azure CLI, PowerShell, ARM Template or Azure Portal below.

- Create a Windows Virtual Machine
- Create a Linux Virtual Machine

Important Note: The Azure Cloud HSM SDK currently only supports Windows (Server 2016, 2019, 2022, 2025) and Linux (Ubuntu 20.04, Ubuntu 22.04, Ubuntu 24.04, RHEL 7, RHEL 8, RHEL 9, CBL Mariner 2). Please ensure the correct OS version when deploying your virtual machine.

Important Note: Customers using any version of Windows Server should install the most recent version of the Visual C++ Redistributable.

Initialize and Configure your Azure Cloud HSM

STEP 9: Download and Install Azure Cloud HSM SDK

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

- **Linux Installation:** Customers can copy the Azure Cloud HSM deb or rpm package to their VM, then use dpkg to install it. The Azure Cloud HSM SDK is installed under /opt/azurecloudhsm. This is the directory for all Azure Cloud HSM management and client utilities including shared libraries for PKCS#11 and OpenSSL.

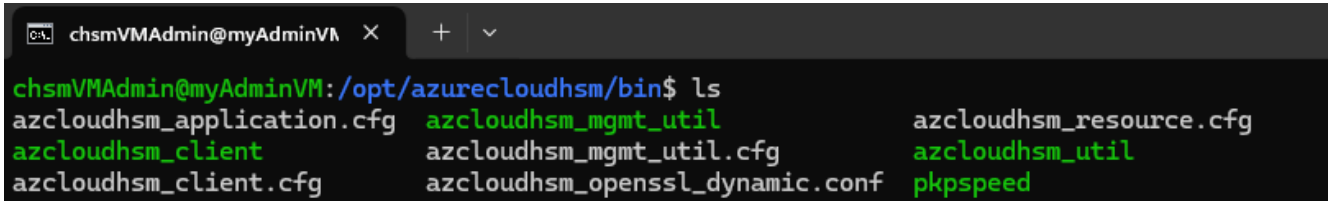
  The example below shows how to download and install the Azure Cloud HSM package. Azure Cloud HSM provides MSI, RPM, and DEB packages tailored to your operating system. Separate packages are available for OpenSSL v1 and OpenSSL v3. It is critical to select the SDK package that matches the OpenSSL version installed on your system. Using a mismatched package may cause the Azure Cloud HSM installation or configuration to fail.

  - wget -p ~/ https://github.com/microsoft/MicrosoftAzureCloudHSM/releases/download/AzureCloudHSM-ClientSDK-2.0.2.3/AzureCloudHSM-ClientSDK-2.0.2.3.deb

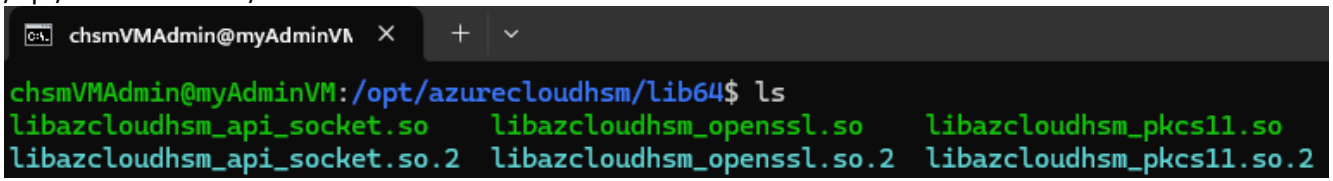
- sudo dpkg --install ~/AzureCloudHSM-ClientSDK-2.0.2.3.deb

Directory Example:

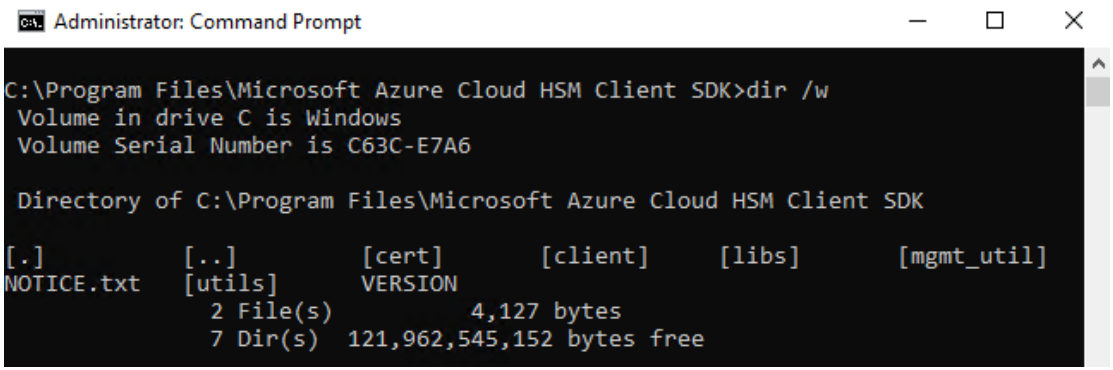
Azure Cloud HSM Management and Client Utilities
/opt/azurecloudhsm/**bin**



Azure Cloud HSM Shared Libraries (PKCS#11 & OpenSSL)
/opt/azurecloudhsm/**lib64**



- **Windows Installation:** To install the Azure Cloud HSM SDK for Windows, download and run the Azure Cloud HSM SDK MSI (Windows Installer Package). This installer will automatically configure all necessary environment variables and dependencies and set up the Cloud HSM Client to run as a service. In the example below, running the Cloud HSM MSI deploys the SDK to C:\Program Files\Microsoft Azure Cloud HSM Client SDK



STEP 10: Configure the Azure Cloud HSM Client Tools

You will need to update the "hostname" property in the azcloudhsm_resource.cfg file. The azcloudhsm_resource.cfg is used to initialize the Azure Cloud HSM cluster. It must point to the private link FQDN for "hsm1" before the cluster is fully initialized as only hsm1 is currently running.

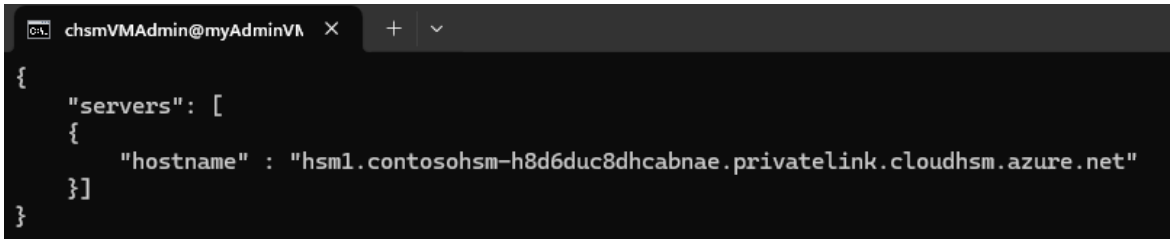
To find the FQDN that includes private link go to your resource group, and select Private Endpoint then select DNS Configuration. You will see configuration name and the FQDN that includes *.privatelink.cloudhsm.azure.net for each HSM instance. Use "hsm1" FQDN as the hostname for the azcloudhsm_resource.cfg file.

Important Note: For Linux there is one file location for azcloudhsm_resource.cfg that you will update under /bin. For Windows there are two file locations for azcloudhsm_resource.cfg that you will need to update under \client and \mgmt_util.

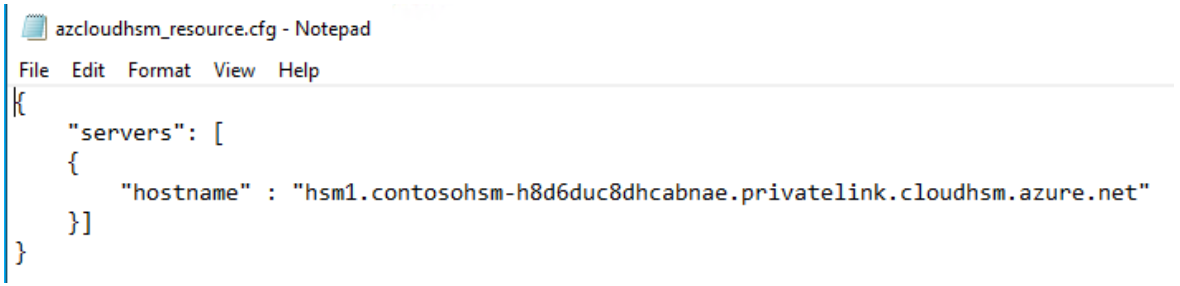
Important Note: The FQDN in the configuration file is a bootstrap endpoint, not a fixed dependency. After connecting, the client can communicate with any healthy HSM node. If the initial node becomes unavailable, the client reconnects to another node and operations continue. Repaired or replaced nodes resume normal participation via the service's self-healing process.

Configuration name	FQDN	IP address
privatelink-cloudhsm-azure-net		
	hsm1.chsm-mycloudhsm-fcdrgvhtfkhxgcgg.privatelink.cloudhsm.azure.net	
		10.0.2.4
	hsm2.chsm-mycloudhsm-fcdrgvhtfkhxgcgg.privatelink.cloudhsm.azure.net	
		10.0.2.5
	hsm3.chsm-mycloudhsm-fcdrgvhtfkhxgcgg.privatelink.cloudhsm.azure.net	
		10.0.2.6

- **Linux Configuration:**
  For Linux, the azcloudhsm_resource.cfg file to be updated is located under /opt/azurecloudhsm/bin. For Linux this single cfg file is used by both the client and mgmt utility to initialize and provide connection reference to your cloud hsm.



- **Windows Configuration:**
  For Windows, you will need to update azcloudhsm_resource.cfg in two locations, \mgmt_util\azcloudhsm_resource.cfg and \client\azcloudhsm_resource.cfg. For Windows, the cfg file under \mgmt_util is used to initialize and provide connection reference to your cloud hsm. While the cfg file under \client is used for connection reference and establishing the client to run as a Win32 service.



Important Note: Customers that want to run the Azure Cloud HSM SDK on Windows will need to install OpenSSL on their server. We recommend installing the Chocolatey Package Manager on your Windows Server to ease with the installation of OpenSSL. Once installed you may need to close your console window and open a new console window for next steps.

Windows Installation for Chocolatey

1. Open PowerShell prompt as 'Administrator'
2. Run the following command to Set Execution Policy and Download Chocolatey:

```
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

3. Run the following command for OpenSSL Installation:

```
choco install openssl --version 1.1.1.1900 -y
```

STEP 11: Take Ownership of your Azure Cloud HSM.

Important Note: Microsoft advises customers to limit access to the POTA private key (PO.key). The AOTA/POTA private keys are equivalent to root access and can reset passwords for crypto officer users in a partition (AOTA for partition 0, POTA for user partitions). The PO.key is unnecessary for HSM access during runtime; it's only required for the initial signing of POAC and CO password resets if necessary. Microsoft suggests storing the PO.key offline and performing the initial POAC signing onto an offline machine, if possible.

Customers are accountable for safeguarding their POTA private key (PO.key). Losing the POTA private key results in the inability to execute CO password recovery. Microsoft advises customers to securely store their POTA private key (PO.key) and maintain suitable backups.

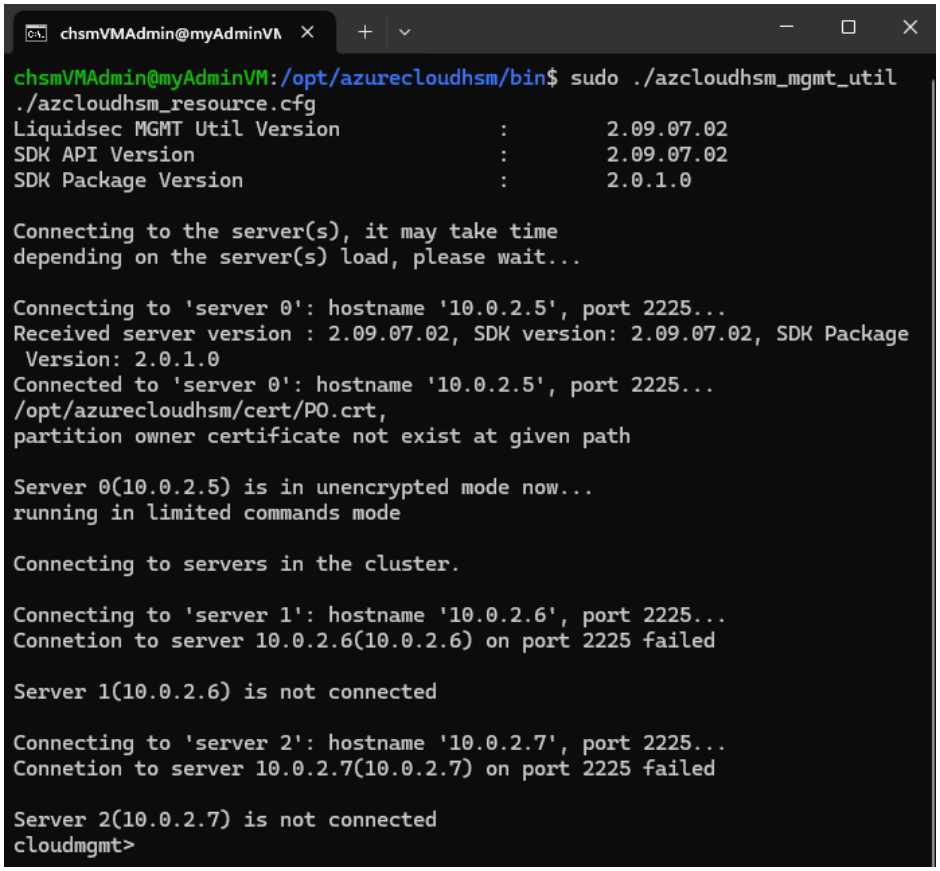
1. Start the azcloudhsm_mgmt_util by executing the following below:

Linux: Use sudo as you will be saving the cert request file under the /opt/azurecloudhsm/cert directory.

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

Windows:

```
cd mgmt_util
.\azcloudhsm_mgmt_util.exe .\azcloudhsm_resource.cfg
```



Important Note: You will observe that only 'server 0' connection shows success after starting azcloudhsm_mgmt_util. This is normal expected behavior as only 'hsm1' is currently running. Only after you complete all the steps provided below will it show all 3 Azure Cloud HSM partitions initialized and connection successful.

2. Inside the management util, the prompt becomes "cloudmgmt". To list the cluster info, execute:

getClusterInfo

```
cloudmgmt>getClusterInfo

Cluster info from server 0(10.0.2.4):
Node Id         Hostname                Port
1               hsm1.keithp-chsm-aycya2e2fsb9fxhu.cloudhsm.azure.net            2225
2               hsm2.keithp-chsm-aycya2e2fsb9fxhu.cloudhsm.azure.net            2225
3               hsm3.keithp-chsm-aycya2e2fsb9fxhu.cloudhsm.azure.net            2225
Server 1 is not connected
Server 2 is not connected
cloudmgmt>
```

3. The following steps need to target only server 0. We can switch to server 0 by executing:

```
server 0
```

The prompt should change from “cloudmgmt” to “server0” as shown below.

```
cloudmgmt>server 0
Server is in 'Unencrypted' mode...
server0>
```

4. Retrieve the CSR (Certificate Signing Request):
Because the client package structure is slightly different for Windows and Linux, the cert should be stored in different relative paths.

Linux:

```
getCertReq ../cert/P1.csr
```

Windows:

```
getCertReq ..\cert\P1.csr
```

```
server0>getCertReq ..\cert\P1.csr
getCertReq success
```

5. Exit the azcloudhsm_mgmt_util. Type ‘exit’ in “server0” prompt, and then type ‘quit’ in “cloudmgmt” prompt.

6. Create the PO key using openssl:

Important Note: Make the PO certificate long lived (~10 years), as expiration can result in permanent loss of administrative access.

Expiration has no impact on runtime cryptographic operations, keys, PKCS#11, JCE, OpenSSL, utilities or applications those will continue to function normally. The PO certificate is used solely for administrative authentication. If it expires, cryptographic workloads remain unaffected, but administrative access to the HSM may be lost.

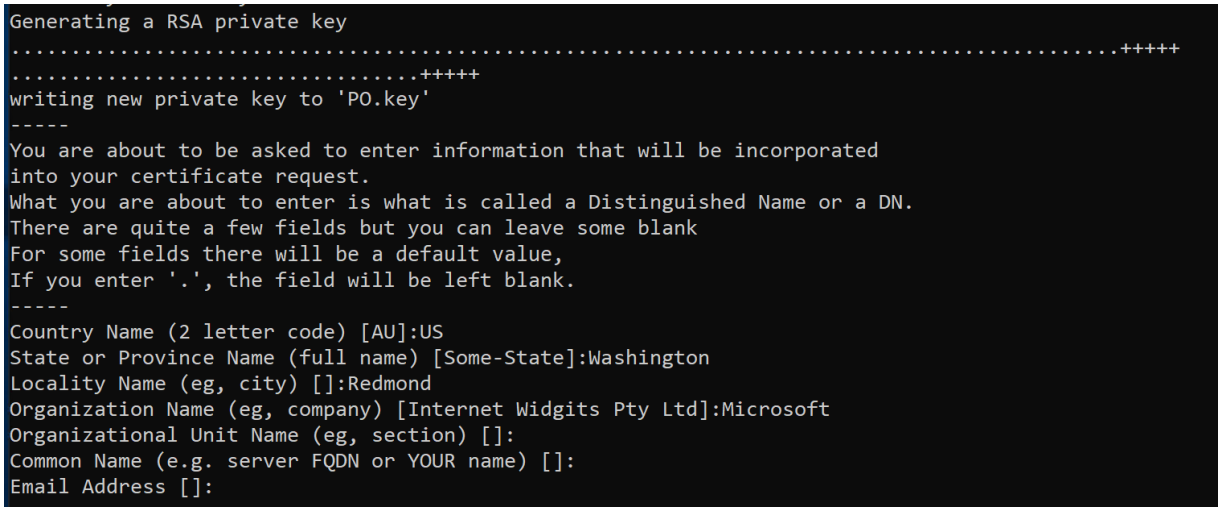
Linux:

```
cd /opt/azurecloudhsm/cert
sudo openssl req -newkey rsa:2048 -nodes -keyout PO.key -x509 -days 3650 -out PO.crt
```

Windows: You will need to run the following command from PowerShell under the /cert directory where you retrieved the CSR too.

```
cd cert
openssl req -newkey rsa:2048 -nodes -keyout PO.key -x509 -days 3650 -out PO.crt
```

- You will be asked to enter information that will be incorporated into your certificate request.



7. Sign the partition cert with the PO key:

Linux:

```
sudo openssl x509 -days 3650 -req -in P1.csr -CA PO.crt -CAkey PO.key -set_serial 01 -out POAC.crt
```

Windows:

openssl x509 -days 3650 -req -in P1.csr -CA PO.crt -CAkey PO.key -set_serial 01 -out POAC.crt

```
chsmVMAdmin@myLinuxVM:/usr/local/bin/AzureCloudHSM-ClientSDK-2.0.0.0/cert$ sudo openssl x509 -days
 365 -req -in P1.csr -CA PO.crt -CAkey PO.key -set_serial 01 -out POAC.crt
Signature ok
subject=C = US + ST = CA + O = Cavium + OU = N3FIPS + L = SanJose, CN = "HSM:CF7B651FCF0C7DF3E2D08
C6F6F8A6A:PARTN:22, for non-FIPS mode"
Getting CA Private Key
```

Note the path to the cert directory is also present in the 2 configuration files. That's why we performed the cert operation in that directory. If you chose to use a different directory, then the .cfg files mentioned in the previous section would need to be updated accordingly.

8. Now we are ready to store the certs in HSM. Go back to the azcloudhsm_mgmt_util tool, and execute the following:

Linux:

```
sudo ./azcloudhsm_mgmt_util ./azcloudhsm_resource.cfg
server 0
loginHSM PRECO admin tenantadmin
storeCert ../cert/PO.crt 4
storeCert ../cert/POAC.crt 8
```

Windows:

```
.\azcloudhsm_mgmt_util.exe .\azcloudhsm_resource.cfg
server 0
loginHSM PRECO admin tenantadmin
storeCert ..\cert\PO.crt 4
storeCert ..\cert\POAC.crt 8
```

```
server0>storeCert ..\cert\PO.crt 4
storeCert success
server0>storeCert ..\cert\POAC.crt 8
storeCert success
logging out
E2E enabled on server 0(10.0.2.4)
server0>
```

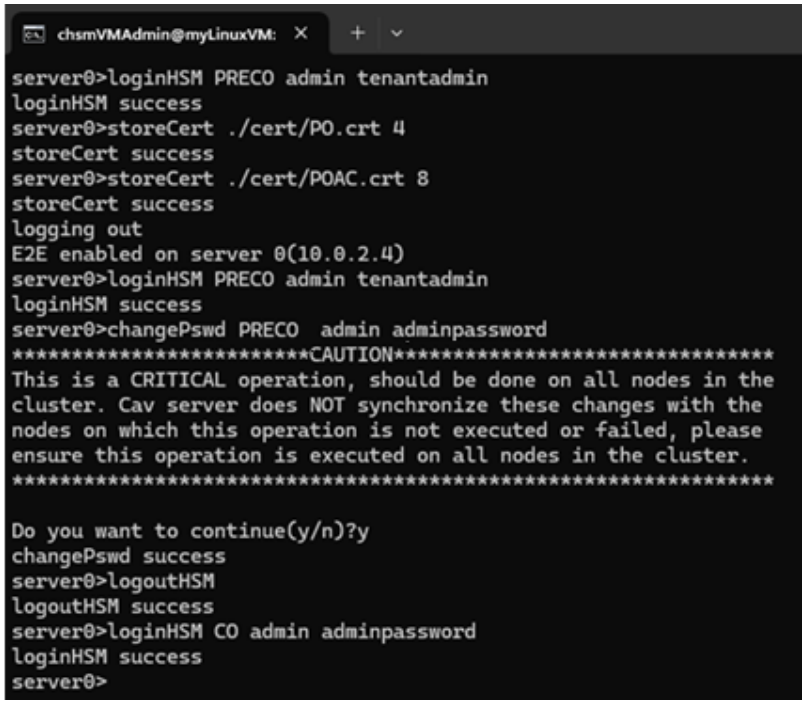
9. Promote PRECO to CO, and login as CO.

Important Note: We strongly recommend that you choose a unique, strong password different from the examples provided below. The passwords in this guide are for demonstration purposes only and should not be used in practice.

Important Note: Please secure your HSM user credentials. Ensuring the secure protection of your HSM user credentials is paramount, as these credentials grant access to perform cryptographic and management operations on your HSM. Azure Cloud HSM does not retain access to your HSM user credentials. Therefore, if access is lost, Microsoft cannot aid if you lose access to your credentials.

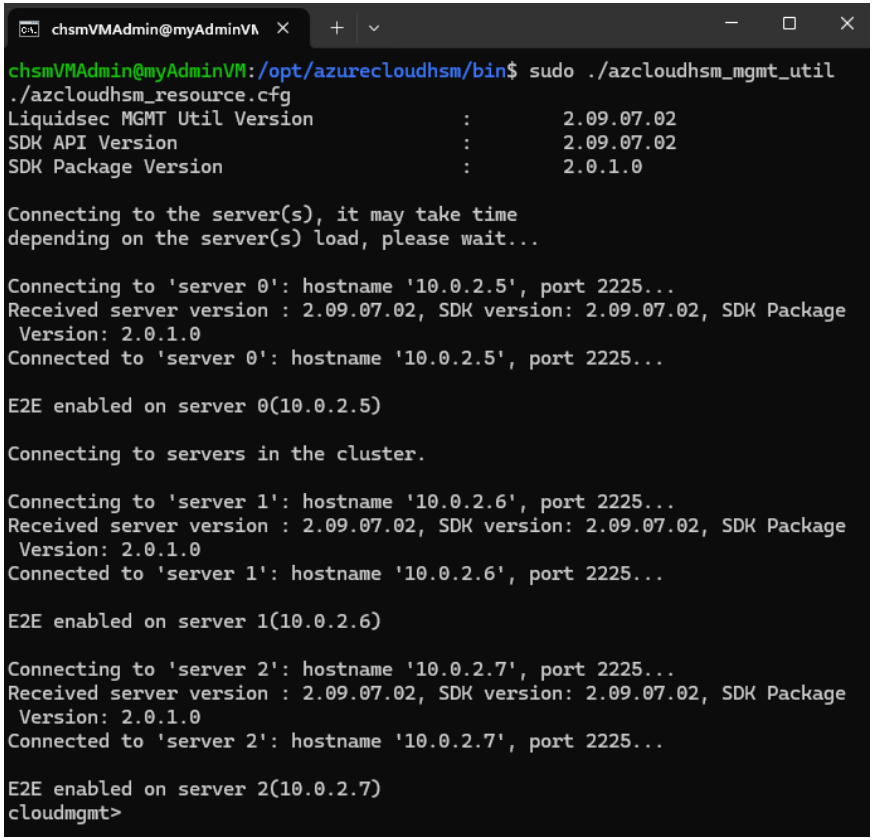
The Pre-Crypto Officer (PRECO) is a factory-default administrative user created during partition initialization. Changing the default PRECO password promotes the account to a Crypto Officer (CO). This is a required initialization step, along with uploading the Partition Owner certificate (PO.crt).

```
loginHSM PRECO admin tenantadmin
changePswd PRECO admin adminpassword
logoutHSM
loginHSM CO admin adminpassword
```



10. Wait a few minutes for the backup restore to complete in the background. You can confirm by checking the activation status of your Azure Cloud HSM within the Azure Portal as well as restarting the azcloudhsm_mgmt_util. It should be able to connect to all 3 HSMs after this operation is done.

This final step completes your deployment and initialization of an Azure Cloud HSM cluster.



Quickstart: Performing HSM Operations

azcloudhsm_mgmt_util

Creating a 'user' with the Crypto User role

This example shows starting azcloudhsm_mgmt_util and administrator logging on as CO, then proceeds to create 'cu1' user with crypto user role. Do not attempt to create a new user if all 3 HSMs in your Cloud HSM cluster have not completed initialization.

Important Note: We strongly recommend that you choose a unique, strong password different from the examples provided below. The passwords in this guide are for demonstration purposes only and should not be used in practice.

Important Note: Please secure your HSM user credentials. Ensuring the secure protection of your HSM user credentials is paramount, as these credentials grant access to perform cryptographic and management operations on your HSM. Azure Cloud HSM does not retain access to your HSM user credentials. Therefore, if access is lost, Microsoft cannot aid if you lose access to your credentials.

Linux:

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

Windows:

```
.\azcloudhsm_mgmt_util.exe .\azcloudhsm_resource.cfg
loginHSM CO admin adminpassword
createUser CU cu1 user1234
logoutHSM
loginHSM CU cu1 user1234
```

```
cloudmgmt>loginHSM CO admin adminpassword
loginHSM success on server 0(10.0.2.4)
loginHSM success on server 1(10.0.2.5)
loginHSM success on server 2(10.0.2.6)
cloudmgmt>createUser CU cu1 user1234
**********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
Creating User cu1(CU) on 3 nodes
createUser success on server 0(10.0.2.4)
createUser success on server 1(10.0.2.5)
createUser success on server 2(10.0.2.6)
cloudmgmt>logoutHSM
logoutHSM success on server 0(10.0.2.4)
logoutHSM success on server 1(10.0.2.5)
logoutHSM success on server 2(10.0.2.6)
cloudmgmt>loginHSM CU cu1 user1234
loginHSM success on server 0(10.0.2.4)
loginHSM success on server 1(10.0.2.5)
loginHSM success on server 2(10.0.2.6)
cloudmgmt>
```

azcloudhsm_client

Running azcloudhsm_client as a service

We recommend running azcloudhsm_client as a service in a production environment since it needs to be continuously operational for the client-server model to establish connections and execute cryptographic operations. Manual execution of the azcloudhsm_client should be for development and testing purposes only.

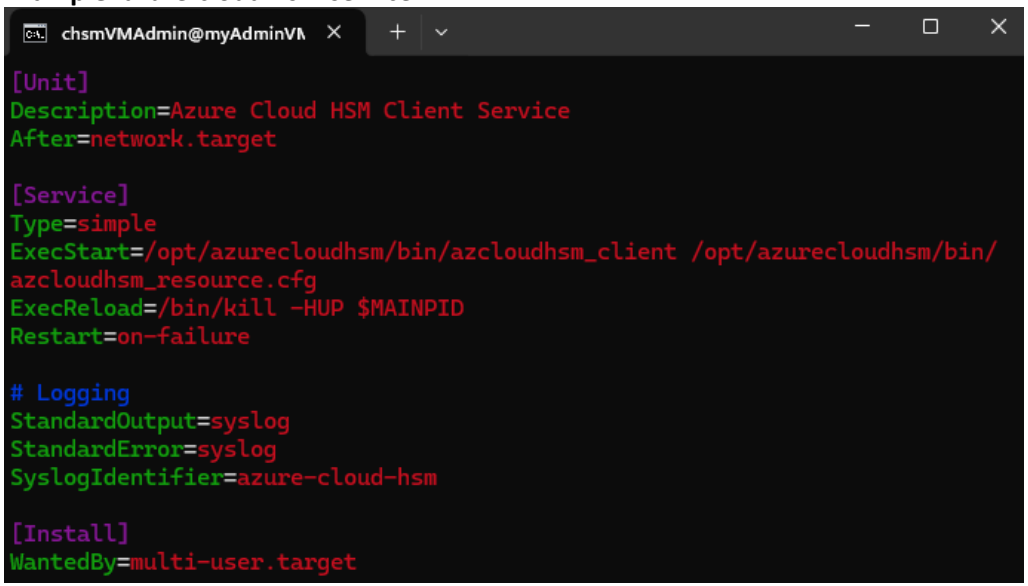
Linux (Recommended):

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 continuous 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 &

Windows (Recommended):

If you installed the Azure Cloud HSM SDK using MSI, the client is already configured to run as a service. If the client is not running, you can open Services.msc, right-click on the Microsoft Azure Cloud HSM Client Service, and select "Start."

Windows (Manual):

Start the client daemon if it is not running.

```
cd C:\Program Files\Microsoft Azure Cloud HSM Client SDK
.\azcloudhsm_client.exe .\azcloudhsm_resource.cfg
```

azcloudhsm_util

The following HSM operations examples below will require that azcloudhsm_client service is running. Within the examples below you will login in as 'CU' to your Cloud HSM. The 'CU' user is the same user as you created in the above example.

Important Note: Syntax for azcloudhsm_util commands below can be found by simply executing any supported command with -h for help.

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.*

Linux:

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

Windows:

```
.\azcloudhsm_util.exe
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.*

Linux:

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

Windows:

```
.\azcloudhsm_util.exe
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.

   Linux:

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

   Windows:

   ```
   .\azcloudhsm_mgmt_util.exe .\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
```

Generate an RSA Key Pair

The example below shows how to use **genRSAKeyPair** to create asymmetric key pairs in your Cloud HSM. This azcloudhsm_util command creates an RSA key pair with a 2048-bit modulus and an exponent of 65537.

genRSAKeyPair -m 2048 -e 65537 -l rsa-key-1

Generate an ECC Key Pair

The example below shows how to use **genECCKeyPair** to create an ECC key pair in your Cloud HSM. This azcloudhsm_util command uses an NID_sect571r1 elliptic curve and an ecc14 label to create an ECC key pair.

genECCKeyPair -i 14 -l ecc14

Generate an AES Key

The example below shows how to use **genSymKey** to create symmetric keys in your Cloud HSM. This azcloudhsm_util command creates a 256-bit AES key with an aes256 label.

```
genSymKey -t 31 -s 32 -l aes256
```

Perform Sign Operation

The **sign** command uses a chosen private key to generate a signature. To use **sign**, you must first have a private key in your Cloud HSM. You can generate a private key with the examples above using **genRSAKeyPair**, **genECCKeyPair**, or **genSymKey**. You can also import a private key with the **importPrivateKey** command. The example below shows how to use **sign** to sign a file. This command signs a file named hash.bin with a private key with a handle 262159. It uses the SHA256_RSA_PKCS (1) signing mechanism and saves the resulting signed file as sign.bin.

```
sign -f hash.bin -k 262159 -m 1 -out sign.bin
```

Perform Verify Operation

The **verify** command confirms whether a signed file was generated from a source file using the corresponding private key. It compares the signed file against the source file and validates the signature cryptographically using a **public key** and the specified signing mechanism.
The example below shows how to verify that sign.bin is a valid signature for hash.bin using public key 262158 with the SHA256_RSA_PKCS signing mechanism.

```
verify -f hash.bin -s sign.bin -k 262158 -m 1
```

Find Key

The **findKey** command is used to search for keys by the values of the key attributes. When a key matches all the criteria that you set, **findKey** returns the key handle. With no parameters, **findKey** returns the key handles of all the keys that you can use in your Cloud HSM. Like all commands, **findKey** is user specific. It returns only the keys that the current user can use. This includes keys that the current user owns and keys that have be shared with the current user. The example and command below show how to use **findKey** to find and identify all keys in your Cloud HSM for the current user.

```
findKey
```

Find Single Key

The **findSingleKey** command verifies that a key exists on all partitions of your Cloud HSM cluster. The example and command below verify that key 262158 exists on all three partitions in your Cloud HSM cluster.

```
findSingleKey -k 262158
```

Delete Key

The **deleteKey** command deletes a key from the Cloud HSM. You can only delete one key at a time. Deleting one key in a key pair has no effect on the other key in the pair. Only the key owner can delete a key. Users who share the key can use it, but not delete it. The example below shows how to use **deleteKey** to delete keys from your Cloud HSM. This command deletes the key with key handle 262160.

```
deleteKey -k 262160
```

APPENDIX

Frequently Asked Questions

- **Azure Cloud HSM shows FIPS STATE 0. What does that mean?**
  Azure Cloud HSM is FIPS 140-3 Level 3 certified. The 'STATE' refers to a restrictive policy that limits the cryptographic algorithms permitted on the HSM. This specific policy is not supported. In State 0, Azure Cloud HSM imposes no restrictions on the cryptographic functions available to customers.

- **Can I update the Partition Owner Cert once it has been uploaded.**
  No. Users cannot change the partition owner cert once it is put in place. If you uploaded the PO.crt in error you will need to delete your Cloud HSM resource and deploy again.

- **Why am I getting a *Certificate ERROR: [certificate signature failure]* when running azcloudhsm_mgmt_util?**
  This error commonly occurs when a customer created and initialized their Azure Cloud HSM from another VM and attempts to install the Cloud HSM Client SDK and run from another VM that is missing or does not have the correct PO.crt from the Admin VM you initialized from. If you copy the PO.crt from your Admin VM to your new VM and rerun the azcloudhsm_mgmt_util you should see a successful connection to your HSM.
  - If you get 'permission denied' trying to copy over the PO.crt perform the following to confirm the PO.crt is present and retry running azcloudhsm_mgmt_util again.
    - cd /var/tmp
    - sudo cp /opt/azurecloudhsm/cert/PO.crt .
    - sudo chmod 777 PO.crt
    - sudo scp ./PO.crt username@ipaddress:<filepath>

- **Why am I getting an *INF: shutdown_ssl_socket: SSL_shutdown sent close_notify alert* when running azcloudhsm_client?**
  This error commonly occurs when a customer created and initialized their Azure Cloud HSM from another VM and attempts to install the Cloud HSM Client SDK and run from another VM that is missing or does not have the correct PO.crt from the Admin VM you initialized from. If you copy the PO.crt from your Admin VM to your new VM and rerun the azcloudhsm_client you should see a successful connection to your HSM.

- If you get 'permission denied' trying to copy over the PO.crt perform the following and confirm the PO.crt is present and retry running azcloudhsm_client again.
  - cd /var/tmp
  - sudo cp /opt/azurecloudhsm/cert/PO.crt .
  - sudo chmod 777 PO.crt
  - sudo scp ./PO.crt username@ipaddress:<filepath>

Azure Cloud HSM SDK Specifications

The Azure Cloud HSM SDK contains client-side tools and utilities. The utilities below are provided to the end user (partition owner) for performing management and cryptographic operations on their Azure Cloud HSM partitions using PCO and PCU.

Management Utility (azcloudhsm_mgmt_util)

The Management Utility is an executable which takes in a configuration file (azcloudhsm_resource.cfg) containing HSM hostname endpoint, while the file (azcloudhsm_mgmt_util.cfg) contains details like client ssl cert location, POAC cert path, etc.

The azcloudhsm_mgmt_util has the following features/usage:

- Used for claiming an allocated partition and establishing an end-to-end channel.
- Used for administrative operations on a partition.

There are two scopes the azcloudhsm_mgmt_util supports: cloudmgmt (global) and server[x] which is partition specific.

CloudMgmt Global Scope Commands:

azcloudhsm_mgmt_util is like an aggregate prompt. Commands execute on all the partitions sequentially. If a command fails for any of the partitions in the Cloud HSM cluster, the user is notified of an error.

Note: If any of the commands which change the state of the partitions (like createUser, setPolicy, modifyKeyOwner, etc.) fail on certain partitions because those endpoints were unreachable, those changes will be synced back across the partitions later by the Azure Cloud HSM service when the server corresponding to that partition comes up again.

CloudMgmt Help:

approveToken	getTokenTimeout	resetUserAuthPubKey
changePswd	getUserHash	server
createUser	getUserInfo	setAttribute

deleteUser	help	setAuditUDD
delToken	info	setConfigPreGenCacheSize
deregisterUserAuthPubKey	listAttributes	setHSMConfig
disableUserAccess	listKBKSlots	setKBKPrimary
enable_e2e	listTokens	setMValue
findAllKeys	listUserAuthPubKeys	setPolicy
getAttribute	listUsers	setTokenTimeout
getAuditUDD	loginHSM	setUserAttributes
getChallenge	getKeyInfo	setUserAuthPubKey
getClusterInfo	logoutHSM	shareKey
getConfigPreGenCacheSize	modifyKeyOwner	smAppCtrl
getHSMInfo	quit	smAppDelete
getKeyDiffMap	registerMofnPubKey	smAppDiagInfo
getMValue	registerQuorumPubKey	smAppUpdate
getPolicy	registerUserAuthPubKey	storeFixedKey
getPreGenCacheSize	renamePCO	unlockCO
getSingleUserHash		unlockUser
getToken		updateUserAuthPubKey

Server[X] Scope Commands:

These commands are directed towards one of the partitions via its corresponding endpoint. While azcloudhsm_mgmt_util is running at the cloudmgmt prompt type ‘server 0’ and enter.

Server[X] Help:

approveToken	getToken	setConfigPreGenCa
backupPartition	getTokenTimeout	SAFE_STATE cheSize
changePswd	getUserDiffMap	setHSMConfig
clone	getUserHash	setKBKPrimary
createUser	getUserInfo	setMValue
deleteUser	help	setPolicy
delToken	listAttributes	setTokenTimeout
deregisterUserAuthPubKey	listKBKSlots	setUserAttributes
disableUserAccess	listTokens	setUserAuthPubKey
enable_e2e	listUserAuthPubKeys	shareKey
exit	listUsers	smAppCtrl
findAllKeys	loginHSM	smAppDelete
generateKEK	logoutHSM	smAppDiagInfo
getAttribute	modifyKeyOwner	smAppUpdate
getAuditUDD	reconnect	storeCert
getCert	registerMofnPubKey	storeFixedKey
getCertReq	registerQuorumPubKey	syncKey
getChallenge	registerUserAuthPubKey	syncUser
getClusterInfo	renamePCO	unlockCO
getConfigPreGenCacheSize	resetUserAuthPubKey	unlockUser
getHSMInfo	restorePartition	updateUserAuthPubKey
getKeyDiffMap	setAttribute	
getKeyInfo	setAuditUDD	
getMValue		
getPartitionInfo		
getPolicy		
getPreGenCacheSize		
getSingleUserHash		

azcloudhsm_util (azcloudhsm_util.exe)

azcloudhsm_util allows you to directly execute operations on your HSM over the E2E channel established by the client daemon (azcloudhsm_client). azcloudhsm_util communicates directly with the client daemon which takes its request to the HSM partitions over the established E2E channel. The client daemon uses PCU credentials to communicate with the HSM. Only PCU scoped commands/operations can be performed using azcloudhsm_util.

Command	Description
exit	Exits application
help	Displays this information
Audit Log Commands:	
getAuditUDD	Get the audit UDD
setAuditUDD	Set/Clear the audit UDD
Certificate Setup Commands:	
getCert	Gets Partition Certificates stored on HSM
Configuration and Admin Commands:	
getHSMInfo	Gets the HSM Information
getPartitionInfo	Gets the Partition Information
listUsers	Lists all users of a partition
loginHSM	Login to the HSM
loginStatus	Gets the Login Information
logoutHSM	Logout from the HSM
Context based user authorization Commands:	
allocContext	Allocate Context
authUser	Authorize User
freeContext	Free Context
Helper Commands:	
Error2String	Converts Error codes to Strings saving key handle in fake PEM format
getCaviumPrivKey	Saves an RSA private key handle in fake PEM format.
IsValidKeyHandlefile	Checks if private key file has an HSM key handle or a real key
listAttributes	List all attributes for getAttributes
listECCCurveIds	List HSM supported ECC CurveIds
Key Derivation Commands:	
deriveSymKey	Derive Symmetric Key
Key Generation Commands:	
Asymmetric Keys:	
genDSAKeyPair	Generates a DSA Key Pair
genECCKeyPair	Generates an ECC Key Pair

genRSAKeyPair	Generates an RSA Key Pair
Symmetric Keys:	
genParkingKey	Generates a parking key
genPBEKey	Generates a PBE DES3 key
genSymKey	Generates a Symmetric keys
splitKey	Split one symmetric key into multiple symmetric keys
Key Import/Export Commands:	
atomicGenAndWrapASymKey	Generates Asymmetric key and Wraps its private key with provided public key(DER format)
atomicGenAndWrapSymKey	Generates Symmetric key and Wraps with provided public key(DER format)
createPublicKey	Creates an RSA public key
exportPrivateKey	Exports RSA/DSA/EC private key
exportPubKey	Exports RSA/DSA/EC Public key
exSymKey	Exports a Symmetric key
importPrivateKey	Imports RSA/DSA/EC private key
importPubKey	Imports RSA/DSA/EC Public key
imSymKey	Imports a Symmetric key
unWrapKey	UnWraps a key into HSM using the specified handle
unWrapKeyWithSize	UnWraps a key into HSM using the key size
wrapKey	Wraps a key from HSM using the specified handle
Key Management Commands:	
deleteKey	Delete Key
findKey	Find Key
findSingleKey	Find single Key
getAttribute	Reads an attribute from an object
getKeyInfo	Get Key Info about shared users/sessions
setAttribute	Sets an attribute of an object
Key Transfer Commands:	
extractMaskedObject	Extracts a masked object
insertMaskedObject	Inserts a masked object
parkObject	Park an object using a parking key
unparkObject	Unpark a parked object using parking key
M of N commands:	
approveToken	Approves an MxN service

delToken	delete Token(s)
getToken	Initiate an MxN service and get Token
listTokens	List all Tokens in the current partition
Management Crypto Commands:	
aesUnwrapPkcs5Buffer	Does KW PKCS5 Pad Unwrap
aesWrapUnwrap	Does NIST AES Wrap/Unwrap
sign	Generates a signature
verify	Verifies a signature

Algorithms Supported

Azure Cloud HSM supports various cryptographic algorithms. Only the algorithms, modes/methods, and key lengths/curves/moduli supported by its hardware security modules are listed in the following tables.

Supported Non-FIPS Algorithms

Azure Cloud HSM and its hardware security modules support the following Non-FIPS approved algorithms.

Algorithm	**Use/Function**
AES (non-compliant)	Key wrap (TR31/TR34/AES-CBC/AES-GCM, wrap/unwrap), DecimalTable/Data/PIN Encryption/Decryption. FF1/FF3-1 Data Encryption/Decryption • AES GCM supports the IV length from 1 byte to 16 bytes
DES	Derive unique key per transaction (DUKPT) EMV key derivation. Derive PIN from Offset Derive Offset from PIN PIN Verification PVV generation and Verification CVV generation and verification Export Symmetric key/Export Asymmetric key pair using TR31 wrapping. Import/Export using TR34. Import Decimal Table EMV script. EMV ARQC/ARPC

	Data/PIN encryption/decryption
DES MAC	MAC generation and Verification
Double-DES	Derive unique key per transaction (DUKPT), EMV key derivation. Derive PIN from Offset Derive Offset from PIN PIN Verification PVV generation and Verification CVV generation and verification Export Symmetric key/Export Asymmetric key pair using TR31 wrapping. Import/Export using TR34 Import Decimal Table EMV script. EMV ARQC/ARPC Data/PIN encryption/decryption
EC-AES	EC-AES wrap/unwrap (EC BYOK)
ECDH KDF	Key derivation using ECDH followed by HMAC/CMAC counter KDF
ECDSA (non-compliant)	Key generation, Sign, Verify P192, Secp192k1, brainpoolP160r1, brainpool192r1, K-163 and B-163 (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512)
EDDSA (non-compliant)	Key generation, Sign, Verify
KAS-ECC (non-compliant)	EC Key generation and ECDH Curve25519 (128 bits), Curve448 (224 bits)

PBE	Key generation
RSA (non-compliant)	TR34 Import TR34 Export PIN block decryption BYOK Encrypt/Decrypt Asymmetric key encapsulation and un-encapsulation using PKCS#1-v1.5 padding with modulus size 2048, 3072 and 4096 bits Key generation, Sign, Verify (1024-bit)
Shamir's Key Share	Key share
Triple-DES (non-compliant)	Derive unique key per transaction (DUKPT) EMV key derivation. Derive PIN from Offset Derive Offset from PIN PIN Verification PVV generation and verification CVV generation and verification Export Symmetric key/Export Asymmetric key pair using TR31 wrapping Import/Export using TR34 Import Decimal Table EMV script. EMV ARQC/ARPC Data/PIN encryption/decryption

Algorithm	**Caveat**	**Use/Function**
AES	Cert A1947, Key unwrapping. Per IG D.G.	Legacy Key unwrap only • ECB mode: Decrypt; 128, 192, and 256-bit

		• CBC mode: Decrypt; 128, 192, and 256-bit
AES	Cert A1948, Key unwrapping. Per IG D.G	Legacy Key unwrap only • ECB mode: Decrypt; 128, 192, and 256-bit • CBC mode: Decrypt; 128, 192, and 256-bit
EC Diffie-Hellman with non-NIST recommended curves	Cert A1947, Provides between 112 and 256 bits of encryption strength. Per IGs D.F and C.A.	EC-DH Secp224k1(112 bits), Secp256K1 (128 bits) brainpoolP224r1(112 bits), brainpoolP256r1(128 bits), brainpoolP320r1(160 bits), brainpoolP384r1(192 bits), brainpoolP512r1(256 bits) FRP256v1 (128 bits) • Prime order curve, generated as per FIPS 186-4 Section 6.1.1 (SHA-1*, SHA2-224, SHA2-256, SHA2-384, SHA2-512)
ECDSA with non-NIST recommended curves	Cert A1947, Provides between 112 and 256 bits of encryption strength. Per IG C.A.	EC Key generation, sign Secp256K1 (128 bits) brainpoolP224r1(112 bits), brainpoolP256r1(128 bits), brainpoolP320r1(160 bits), brainpoolP384r1(192 bits), brainpoolP512r1(256 bits) FRP256v1 (128 bits) • Prime order curve, generated as per FIPS 186-4 Section 6.1.1 (SHA-1*, SHA2-224, SHA2-256, SHA2-384, SHA2-512)
SHA-1	No security claimed per IG 2.4.A	Fingerprints
Triple-DES SP 800-38B	No security claimed per IG 2.4.A	Fingerprints Key Sizes

		• 192-bit (Generation, Verify)

Supported NIST FIPS Approved Algorithms

Azure Cloud HSM and its hardware security modules support the following NIST FIPS approved algorithms.

Important Note: All symmetric key sizes represent the key strength.

CAVP Cert	Algorithm and Standard	Mode/Method	Description/ Key Size(s) / Key Strength(s)	Use / Function
A1946	RSA SigVer (FIPS186-4)	RSA SigVer (FIPS186-4)	RSA 2048 with SHA2-256 Uses SHA2-256 (FIPS 180-4) (#A1946) as the underlying digest algorithm	Firmware integrity verification by U-boot
A1946	SHA2-256 (FIPS 180-4)	SHA2	SHA 256	Firmware integrity verification by U-boot
A1947	AES-CBC (SP 800-38A)	AES	Encrypt/Decrypt: 128, 192, and 256-bit	Data encryption, decryption, key-wrap and key-unwrap
A1947	AES-CCM (SP 800-38C)	AES	Encrypt/Decrypt: 128, 192, and 256-bit Uses AES-ECB (SP 800-38A) (#A1947) as the underlying block cipher	Authenticated Data encryption, decryption
A1947	AES-CMAC (SP 800-38B)	AES	CMAC generate and verify: 128, 192, and 256-bit Uses AES-CBC (SP 800-38A) (#A1947) as the underlying block cipher	Message authentication code generation and verification
A1947	AES-CTR (SP 800-38A)	AES	Encrypt/Decrypt: 128, 192, and 256-bit Uses AES-ECB (SP 800-38A) (#A1947) as the underlying block cipher	Data encryption, decryption
A1947	AES-ECB (SP 800-38A)	AES	Encrypt/Decrypt: 128, 192, and 256-bit	Data encryption, decryption, key-wrap and key-unwrap
A1947	AES-GCM (SP 800-38D)	AES	Encrypt/Decrypt: 128, 192, and 256-bit Uses AES-ECB (SP 800-38A) (#A1947)	Authenticated Data encryption, decryption, key wrap, and key unwrap

			as the underlying block cipher	
A1947	AES-GMAC (SP 800-38D)	AES	GMAC generation: 128, 192, and 256-bitUses AES-ECB (SP 800-38A) (#A1947) as the underlying block cipher	Message Authentication
A1947	ECDSA SigGen (FIPS186-4) (CVL)	ECDSA SigGen (FIPS186-4)	SigGen Component: P-224, P-256, P-384, P-521 (SHA-256, 384, and 512) Uses Hash DRBG (SP 800-90Ar1) (#A1947) as underlying Random Generator P-224, P-256, P-384, and P-521 curves providing 112, 128, 192, or 256 bits of encryption strength respectively	Signature generation
A1947	Hash DRBG (SP 800-90Ar1)	Hash DRBG	SHA-512 based with security strength of 256-bit. No prediction resistance Uses SHA2-512 (#A1947) as the underlying digest algorithm	Random number generation for user, internal IVs, and salt
A1947	HMAC-SHA2-256 (FIPS 198-1)	HMAC	HMAC-SHA2-256 Uses SHA2-256 (#A1947) as the underlying digest algorithm	MAC generation, verify, KAS and KDF
A1947	HMAC-SHA2-384 (FIPS 198-1)	HMAC	HMAC-SHA2-384 Uses SHA2-384 (#A1947) as the underlying digest algorithm	MAC generation, verify, KAS and KDF
A1947	HMAC-SHA2-512 (FIPS 198-1)	HMAC	HMAC-SHA2-512 Uses SHA2-512 (#A1947) as the underlying digest algorithm	MAC generation, verify, KAS and KDF
A1947	KAS-ECC-SSC	KAS	ECC CDH staticUnified: P-224, P-256,	Shared secret computation

	SP800-56Ar3		P-384, P-521, P-224, P-256, P384 and P-521 providing 112 bits,128 bits. 192 bits and 256 bits of encryption strength respectively Uses Hash DRBG (SP800-90Ar1) (#A1947) as underlying Random Generator for the ECDSA KeyGen key pair Uses ECDSA KeyVer (FIPS 186-4) (#A1948) and ECDSA KeyGen (FIPS 186-4) (#A2393) underlying verification for the key pair Uses SHA2-256 (FIPS 180-4) (#A1947), SHA2-384 (FIPS 180-4) (#A1947) and SHA2-512 (FIPS 180-4) (#A1947) as underlying digest algorithm	
A1947	KDF TLS (CVL) (SP 800-135r1)	KDF TLS	TLS-KDF (v1.2) v1.2: SHA2-256, SHA2-384, SHA2-512 Uses HMAC-SHA2-256 (FIPS 180-4) (#A1947), HMAC-SHA2-384 (FIPS 180-4) (#A1947) and HMAC-SHA2-512 (FIPS 180-4) (#A1947) as underlying MAC algorithm	TLS handshake
A1947	RSA Decryption Primitive (SP 800-56Br2) (CVL)	RSA Decryption Primitive	2048 bit, 3072 bit and 4096-bit 2048, 3072, and 4096-bit modulus providing 112, 128, and 150 bits of encryption strength respectively.	Decryption primitive
A1947	RSA Signature Primitive (CVL) (FIPS 186-4)	RSA Signature Primitive	2048 bit, 3072 bit and 4096-bit 2048, 3072, and 4096-bit modulus providing 112, 128, and 150 bits of	Signature primitive

			encryption strength respectively.	
A1947	SHA-1 (FIPS 180-4)	SHA-1	SHA-1	Message digest
A1947	SHA2-256 (FIPS 180-4)	SHA2	SHA2256	Message digest
A1947	SHA2-384 (FIPS 180-4)	SHA2	SHA2-384	Message digest
A1947	SHA2-512 (FIPS 180-4)	SHA2	SHA2-512	Message digest
A1947	SHA3-224 (FIPS 202)	SHA3	SHA3-224	Message digest
A1947	SHA3-256 (FIPS 202)	SHA3	SHA3-256	Message digest
A1947	SHA3-384 (FIPS 202)	SHA3	SHA3-384	Message digest
A1947	SHA3-512 (FIPS 202)	SHA3	SHA3-512	Message digest
A1947	SHAKE-128 (FIPS 202)	SHAKE-128	SHAKE-128	Message digest
A1947	SHAKE-256 (FIPS 202)	SHAKE-256	SHAKE-256	Message digest
A1947	TDES-CBC (SP 800-38A)	TDES	3-key Triple-DES decrypt (supports only 192-bit size)	Data decryption * Legacy use only
A1947	TDES-ECB (SP 800-38A)	TDES	3-key Triple-DES decrypt (supports only 192-bit size)	Data decryption * Legacy use only
A1948	AES-CBC (SP 800-38A)	AES	Encrypt/Decrypt, 128 and 256-bit	Data encryption, decryption, and key unwrap
A1948	AES-CMAC (SP 800-38B)	AES	CMAC generate and verify: 128, 192, and 256-bit Uses AES-CBC (SP 800-38A) (#A1948)	MAC generation and verification

			as the underlying block cipher	
A1948	AES-GCM (KTS) (SP800-38D)	SP 800-38D and SP 800-38F. KTS (key wrapping and unwrapping) per IG D.G.	128, 192, and 256- bit keys providing 128, 192, or 256 bits of encryption strength	Data encryption, decryption, key-wrap, and key-unwrap
A1948	AES-GCM (SP 800-38D)	AES	GCM mode: Authenticated encrypt/decrypt; 128, 192, and 256-bitUses AES-CBC (SP 800-38A) (#A1948) as the underlying block cipher	Authenticated data encryption and decryption, key unwrap
A1948	AES-KW (KTS) (SP 800-38F)	SP 800-38F. KTS (key wrapping and unwrapping) per IG D.G.	128, 192 and 256-bit keys providing 128-bit, 192-bit, or 256 bits of encryption strength	Key wrapping/unwrapping
A1948	AES-KW (SP 800-38F)	AES	KW, 128, 192, and 256-bit Uses AES-CBC (#A1948) as underlying block cipher	Key wrapping/unwrapping
A1948	AES-KWP (KTS) (SP 800-38F)	SP 800-38F. KTS (key wrapping and unwrapping) per IG D.G.	128, 192 and 256-bit keys providing 128-bit, 192-bit, or 256 bits of encryption strength	Key wrapping/unwrapping
A1948	AES-KWP (SP 800-38F)	AES	KWP, 128, 192, and 256-bit Uses AES-CBC (#A1948) as underlying block cipher	Key wrapping/unwrapping
A1948	Counter DRBG (SP 800-90Ar1)	Counter DRBG	AES 256 with df No prediction resistance Uses AES-CBC (SP 800-38A) (#A1948) as the underlying block cipher	Random number generation for user, internal IVs, and salt
A1948	ECDSA KeyGen (FIPS186-4)	Key Gen	Key Gen: P-224, P-256, P-384, P-521 Uses Counter DRBG (SP800-90Ar1) (#A1948) as underlying Random Generator	Key generation
A1948	ECDSA KeyVer	ECDSA KeyVer (FIPS186-4)	Key Ver; P-224, P-256, P-384 and P-521	Key Verification

	(FIPS186-4)			
A1948	ECDSA SigGen (FIPS186-4)	ECDSA SigGen (FIPS186-4)	Signature generation: P-224, P-256, P-384, P-521 (SHA2-256, SHA2-384, SHA2-512) Uses Hash DRBG (SP800-90Ar1) (#A1947) as underlying Random generator Uses SHA2-256 (FIPS 180-4) (#A1948), SHA2-384 (FIPS 180-4) (#A1948) and SHA2-512 (FIPS 180-4) (#A1948) as underlying digest algorithm	Signature Generation
A1948	ECDSA SigVer (FIPS186-4)	ECDSA SigVer(FIPS186-4)	Signature verify: P-224, P-256, P-384, P-521 (SHA- 1, SHA2-256, SHA2-384, and SHA2-512) Uses SHA-1 (FIPS 180-4) (#A1948), SHA2-256 (FIPS 180-4) (#A1948) and SHA2-384 (FIPS 180-4) (#A1948) and SHA2-512 (FIPS 180-4) (#A1948) as underlying digest algorithm	Signature Verification
A1948	HMAC-SHA-1 (FIPS 198-1)	HMAC	HMAC-SHA-1 Uses SHA-1 (FIPS 180-4) (#A1948) as underlying digest algorithm	MAC generation, Verify and KDF
A1948	HMAC-SHA2-256 (FIPS 198-1)	HMAC	HMAC-SHA2-256 Uses SHA2-256 (FIPS 180-4) (#A1948) as underlying digest algorithm	MAC generation, verify, KAS, and KDF
A1948	HMAC-SHA2-384 (FIPS 198-1)	HMAC	HMAC-SHA2-384 Uses SHA2-384 (FIPS 180-4) (#A1948) as underlying digest algorithm	MAC generation, verify, KAS, and KDF
A1948	HMAC-SHA2-512 (FIPS 198-1)	HMAC	HMAC-SHA2-512 Uses SHA2-512((FIPS 180-4) (#A1948) as underlying digest algorithm	MAC generation, verify, KAS, and KDF

A1948	KAS-ECC (KAS) (SP 800-56Ar3)	SP 800-56Arev3. KAS-ECC per IG D.F Scenario 2 path (2).	P-521 curve providing 256 bits of encryption strength	Cloning protocol
A1948	KAS-ECC SP800-56Ar3	KAS	ECC CDH Ephemeral Unified P-521 with OneStep KDF (HMAC-SHA2-512) Uses Counter DRBG (#A1948) as underlying Random number Uses ECDSA KeyGen (FIPS186-4) (#A1948) as underlying ECC KeyGen and ECDSA KeyVer (#A1948) as underlying ECDSA Key Verification algorithm.\ Uses SHA2-512) (FIPS 180-4) as underlying digest algorithm	Cloning protocol
A1948	KAS-ECC-SSC SP800-56Ar3	KAS	ECC CDH Static Unified: P-224, P-256, P384, P-521, B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571 P-224, P-256, P384 and P-521 providing 112 bits,128 bits. 192 bits and 256 bits of encryption strength respectively Uses Counter DRBG (SP 800-90Ar1) (#A1948) as underlying Random Generator for the ECDSA KeyGen key pair Uses ECDSA KeyVer (FIPS 186-4) (#A1948) and	Shared secret computation

			ECDSA KeyGen (FIPS 186-4) (#A1948) underlying Verification and generation for the key pair Uses SHA2-256 (FIPS 180-4) (#A1948), SHA2-384 (FIPS 180-4) (#A1948), SHA2-512(FIPS 180-4) (#A1948) as the underlying digest algorithm	
A1948	KAS-IFC-SSC (SP 800-56Br2)	KAS	RSA-based key exchange scheme KAS1 and KAS2 RSA 2048, 3072, and 4096-bit 2048, 3072, and 4096-bit modulus providing 112, 128, and 150 bits of encryption strength respectively Uses Counter DRBG (SP 800-90Ar1) (#A1948) as underlying Random Generator for the RSA Key Pair Generation Uses HMAC-SHA2-256 (#A1948), HMAC-SHA2-384 (#A1948) and HMAC-SHA2-512 (#A1948) as underlying MAC algorithm Uses RSA KeyGen (FIPS	PEK and KLK generation and certificate authentication

			186-4) [#A1948] as underlying RSA key generation algorithm	
A1948	KDA HKDF SP800-56Cr1	KDA HKDF SP800-56Cr1	HKDF Uses Counter DRBG (SP 800-90Ar1) (#A1948) as underlying Random Generator Uses HMAC-SHA2-256 (#A1948), HMAC-SHA2-384 (#A1948) and HMAC-SHA2-512 (#A1948) as underlying MAC algorithm	ECDH key derivation and ECDH key wrap
A1948	KDA OneStep SP800-56Cr1	KDA OneStep SP800-56Cr1	Hash-based KDF Uses Counter DRBG (SP 800-90Ar1) (#A1948) as underlying Random Generator Uses HMAC-SHA2-256 (#A1948), HMAC-SHA2-384 (#A1948) and HMAC-SHA2-512 (#A1948) as underlying MAC algorithm	ECDH key derivation and ECDH key wrap
A1948	KDA TwoStep SP800-56Cr1	KDA TwoStep SP800-56Cr1	HMAC/CMAC-based extract- expand KDFs Uses Counter DRBG (SP	ECDH key derivation and ECDH key wrap

			800-90Ar1) (#A1948) as underlying Random Generator Uses HMAC-SHA2-256 (#A1948), HMAC-SHA2-384 (#A1948) and HMAC-SHA2-512 (#A1948) as underlying MAC algorithm Uses AES-CBC (#A1948) as underlying Block cipher for AES-CMAC	
A1948	KDF ANS 9.63 (CVL) (SP 800-135r1)	KDF ANS 9.63	SHA-2 224, 256, 384, and 512 Uses SHA2-256 (#A1948), SHA2-384 (#A1948) and SHA2-512 (#A1948) as underlying digest algorithm	Key derivation and key agreement schemes
A1948	KDF SP800-108	KDF SP800-108	HMAC-SHA-1, SHA2-224, SHA2-256, 384, and 512 KDF Uses HMAC-SHA-1 (#A1948), HMAC-SHA2-256 (#A1948), HMAC-SHA2-384 (#A1948) and HMAC-SHA2-512 (#A1948) as underlying MAC algorithm	Key derivation
A1948	KTS-IFC (KTS) (SP 800-56Br2)	SP 800-56Brev2. KTS-IFC (key encapsulation and un-encapsulation) per IG D.G.	2048, 3072, or 4096-bit modulus providing 112, 128, or 150 bits of encryption strength	Asymmetric key encapsulation and un-encapsulation
A1948	KTS-IFC (SP 800-56Br2)	KTS	KTS-OAEP-basic RSA 2048, 3072, and 4096-bit Uses counter DRBG (SP800-90Ar1) (#A1948) as underlying Random Generator	Asymmetric key encapsulation and un-encapsulation

			Uses HMAC-SHA2-256 (FIPS 180-4) (#A1948), HMAC-SHA2-384 (FIPS 180-4) (#A1948), HMAC-SHA2-512 (FIPS 180-4) (#A1948) as underlying digest algorithm Uses RSA KeyGen (FIPS 186-4) (#A1948) as underlying key generation algorithm	
A1948	PBKDF (SP 800-132)	PBKDF	HMAC with SHA-1, SHA-2 256, 384, and 512 Uses HMAC-SHA-1 (#A1948), HMAC-SHA2-256 (#A1948), HMAC-SHA2-384 (#A1948) and HMAC-SHA2-512 (#A1948) as underlying MAC algorithm	User credentials storage. Master key derived from PBKDF is not used for deriving keys for data encryption/protection, but instead used only for storing passwords as hashes.
A1948	RSA Decryption Primitive (SP 800-56Br2) (CVL)	RSA Decryption Primitive	Modulus sizes: 2048, 3072, and 4096-bit	RSA key transport
A1948	RSA KeyGen (FIPS186-4)	RSA KeyGen (FIPS186-4)	Key generation: 2048, 3072, and 4096-bit Uses Counter DRBG (SP800-90Ar1) (#A1948) as underlying Random Generator	RSA key generation
A1948	RSA SigGen (FIPS186-4)	RSA SigGen (FIPS186-4)	FIPS 186-4 PKCS #1 1.5 and PSS Sig Gen: 2048, 3096, and 4096-bit (SHA-2 256, 384, and 512) Uses SHA2-256 (FIPS 180-4) (#A1948), SHA2-384 (FIPS 180-4) (#A1948), SHA2-512 (#A1948) as underlying digest algorithm 2048, 3072, and 4096-bit modulus	Signature generation

			providing 112, 128, and 150 bits of encryption strength respectively.	
A1948	RSA SigVer (FIPS186-4)	RSA SigVer (FIPS186-4)	FIPS 186-4 PKCS #1 1.5 and PSS SigVer: 1024, 2048, 3072, and 4096-bit (SHA-1, SHA-2 256, 384, and 512) Uses SHA-1 (FIPS 180-4)(#A1947) SHA2-256 (FIPS 180-4) (#A1948), SHA2-384 (FIPS 180-4) (#A1948), SHA2-512 (#A1948) as underlying digest algorithm 2048, 3072, and 4096-bit modulus providing 112, 128, and 150 bits of encryption strength respectively.	Signature verification User authentication Firmware update verification
A1948	SHA-1 (FIPS 180-4)	SHA-1	SHA-1	Digests, HMAC, and KDFs
A1948	SHA2-256 (FIPS 180-4)	SHA2	SHA2-256	Digests, HMAC, signature generation, and KDFs
A1948	SHA2-384 (FIPS 180-4)	SHA2	SHA2-384	Digests, HMAC, signature generation, and KDFs
A1948	SHA2-512 (FIPS 180-4)	SHA2	SHA2-512	Digests, HMAC, signature generation, and KDFs
A1948	TDES-ECB	TDES	Triple-DES 3-key decrypt.	Prerequisite for TDES-KW. * Legacy use only
A1948	TDES-KW (KTS) (SP 800-38F)	SP 800-38F. KTS (key unwrapping) per IG D.G	192-bit key providing 112-bit encryption strength	Key unwrapping * Legacy use only
A1948	TDES-KW (SP 800-38F)	TDES	TKW 3-key DES decrypt Uses TDES-ECB (#A1948) as underlying block cipher	Key unwrapping * Legacy use only
A2393	ECDSA KeyGen (FIPS186-4)	ECDSA KeyGen (FIPS186-4)	Key Gen: P-224, P-256, P-384, P-521 Uses Hash DRBG (SP800-90Ar1)	Key Generation

			(#A1947) as underlying Random Generator for the ECDSA KeyGen key pair	
A2393	KTS-IFC (KTS) (SP 800-56Br2)	SP 800-56Brev2. KTS-IFC (key encapsulation and un-encapsulation) per IG D.G.	2048, 3072, or 4096-bit modulus providing 112, 128, or 150 bits of encryption strength	Asymmetric key encapsulation and un-encapsulation in hybrid environment
A2393	KTS-IFC (SP 800-56Br2)	KTS	RSA key wrap and unwrap of symmetric keys in KTS-OAEP-Basic padding. 2048, 3072, 4096-bit modulus Uses Counter DRBG (SP800-90Ar1) (#A1948) as underlying Random Generator Uses HMAC-SHA2-256 (FIPS 180-4) (#A1947), HMAC-SHA2-384 (FIPS 180-4) (#A1947), HMAC-SHA2-512 (FIPS 180-4) (#A1947) as underlying digest algorithm Uses RSA KeyGen (FIPS 186-4) (#A2393) as underlying key generation algorithm	Asymmetric key encapsulation and un-encapsulation in hybrid environment
A2393	RSA KeyGen (FIPS186-4)	RSA KeyGen (FIPS186-4)	Key generation: 2048, 3072, and 4096-bit Uses Hash DRBG (SP800-90Ar1) (#A1947) as underlying Random Generator for the ECDSA KeyGen key pair	RSA key generation in hybrid environment
KAS-ECC-SSC SP800-56Ar3 (#A1947) and KDF TLS (CVL) (#A1947)	KAS TLS (SP 800-56Ar3)	SP 800-56Arev3. KAS-ECC per IG D.F Scenario 2 path (2)	P-224, P-256, P-384, P-521 curves providing 112, 128, 192, or 256 bits of encryption strength	TLS

KAS-ECC-SSC SP800-56Ar3 (#A1948) KDF ANS 9.63 (CVL) (SP 800-135r1) (#A1948) KAS-ECC-SSC	KAS ANS 9.63 (SP 800-56Ar3)	SP 800-56Arev3. KAS-ECC per IG D.F Scenario 2 path (2).	P-224, P-256, P-384, and P-521 curves providing 112, 128, 192, or 256 bits of encryption strength	ECDH key derivation and ECDH-AES key wrap
KAS-ECC-SSC SP800-56Ar3 (#A1948), KDA HKDF SP800-56Cr1 (#A1948)	KAS KDA HKDF (SP 800-56Ar3)	SP 800-56Arev3. KAS-ECC per IG D.F Scenario 2 path (2).	P-224, P-256, P-384, and P-521 curves providing 112, 128, 192, or 256 bits of encryption strength	ECDH key derivation and ECDH-AES key wrap
KAS-ECC-SSC SP800-56Ar3 (#A1948), KDA OneStep SP800-56Cr1 (#A1948)	KAS KDA ONESTEP (SP 800-56Ar3)	SP 800-56Arev3. KAS-ECC per IG D.F Scenario 2 path (2).	P-224, P-256, P-384, and P-521 curves providing 112, 128, 192, or 256 bits of encryption strength	ECDH key derivation and ECDH-AES key wrap
KAS-ECC-SSC SP800-56Ar3 (#A1948), KDA TwoStep SP800-56Cr1 (#A1948)	KAS KDA TWOSTEP (SP 800-56Ar3)	SP 800-56Arev3. KAS-ECC per IG D.F Scenario 2 path (2).	P-224, P-256, P-384, and P-521 curves providing 112, 128, 192, or 256 bits of encryption strength	ECDH key derivation and ECDH-AES key wrap
KAS-IFC-SSC (SP 800-56Br2) (#A1948) KDA HKDF SP800-56Cr1 (#A1948)	KAS-IFC HKDF (SP800-56Br2)	SP 800-56Brev2. KAS-IFC per IG D.F Scenario 1 path (2).	2048-bit, 3072-bit and 4096-bit modulus providing between 112 and 150 bits of encryption strength	PEK and KLK generation and certificate authentication

KAS-IFC-SSC (SP 800-56Br2) (#A1948) KDA OneStep SP800-56Cr1 (#A1948)	KAS-IFC OneStep (SP800-56Br2)	SP 800-56Brev2. KAS-IFC per IG D.F Scenario 1 path (2).	2048-bit, 3072-bit and 4096-bit modulus providing between 112 and 150 bits of encryption strength	PEK and KLK generation and certificate authentication
KAS-IFC-SSC (SP 800-56Br2) (#A1948) KDA TwoStep SP800-56Cr1 (#A1948)	KAS-IFC TwoStep (SP800-56Br2)	SP 800-56Brev2. KAS-IFC per IG D.F Scenario 1 path (2).	2048-bit, 3072-bit and 4096-bit modulus providing between 112 and 150 bits of encryption strength	PEK and KLK generation and certificate authentication
N/A	ENT (P) (SP 800-90B)	N/A	SP 800-90B entropy source	Entropy Source
Vendor Affirmed	CKG SP 800-133Rev2	CKG	Please refer to section 2.3.2 Algorithm Specific Information	Cryptographic Key Generation; SP 800-133Rev2 and IG D.H

Glossary of Terms

MCO	Master Crypto Officer
PCO	Partition Crypto Officer
PRECO	Pre-Crypto Officer – Least privileged default admin on the Partition. Used only for claiming the partition remotely over TLS connection.
AU	Appliance User – Least privileged Partition user
CU	Crypto User – Partition user capable of performing Crypto operations on the partition.
Azure Cloud HSM Service	Azure Cloud HSM Security Server (Cavium Server)
azcloudhsm_client	Azure Cloud HSM Client-side Daemon process
azcloudhsm_mgmt_util	Azure Cloud HSM Client-side management utility

azcloudhsm_resource.cfg	Azure Cloud HSM hostname parameter file for azcloudhsm_client and azcloudhsm_mgmt_util
MARC – Enum 1	Manufacturer Authentication Root Certificate
MAC (HSM Cert) - Enum 2	Manufacturer Authentication Certificate (Signed by MARC) – Installed by Marvell on the HSM card
AOTAC (HSM_OWNER_CERT) - Enum 32	Adapter Owner Trust Anchor Certificate
AOAC – (HSM_CERT_ISSUED_BY_HO) Enum 64	Adapter Owner Authentication Certificate (Signed by AOTAC)
POTAC (PARTITION_OWNER_CERT) – Enum 4	Partition Owner Trust Anchor Certificate
POAC (PARTITION_CERT_ISSUED_BY_PO) – Enum 8	Partition Owner Authentication Certificate (Partition Key Signed by POTAC)
PAC (PARTITION_CERT_ISSUED_BY_HSM) - Enum 16	Partition cert (Partition Key signed by MAC)
PEK	Password Encryption Key - This is generated at the time of initializing a partition.
pkey	Private SSL Key

Recommended Readings on Azure Security Best Practices

- Security best practices for IaaS workloads in Azure
- Enable just-in-time access on Virtual Machines
- Adopt a Zero Trust approach