Encryption

Abstract

Protect your enterprise data whether it is in flight from an on-premises data lake or at rest in cloud storage.

Table of Contents

Get started with Encryption

Privacera Encryption enhances the data security provided by Privacera Access Management and Privacera Discovery.

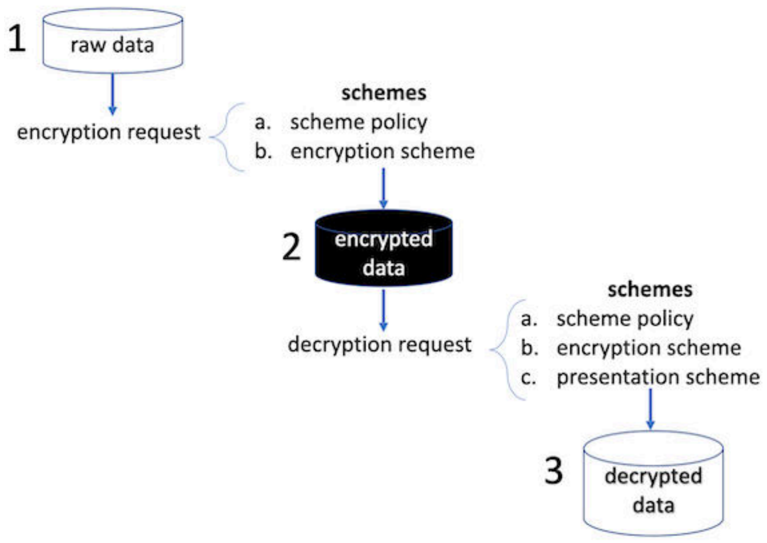
You can encrypt tables, columns, rows, fields, or other data in connected systems. Even if the data are accessible by policies created in Privacera Access Management, the encrypted data cannot be seen.

Encryption can be two-way: you can encrypt the data in place and decrypt it later. Or it can be one-way: with hashing or overwriting with string literals. You can replace the original data to make it invisible and unrecoverable.

You can also completely mask data with a one-way transform.

The encryption process

The following graphic shows the general process of Privacera Encryption.

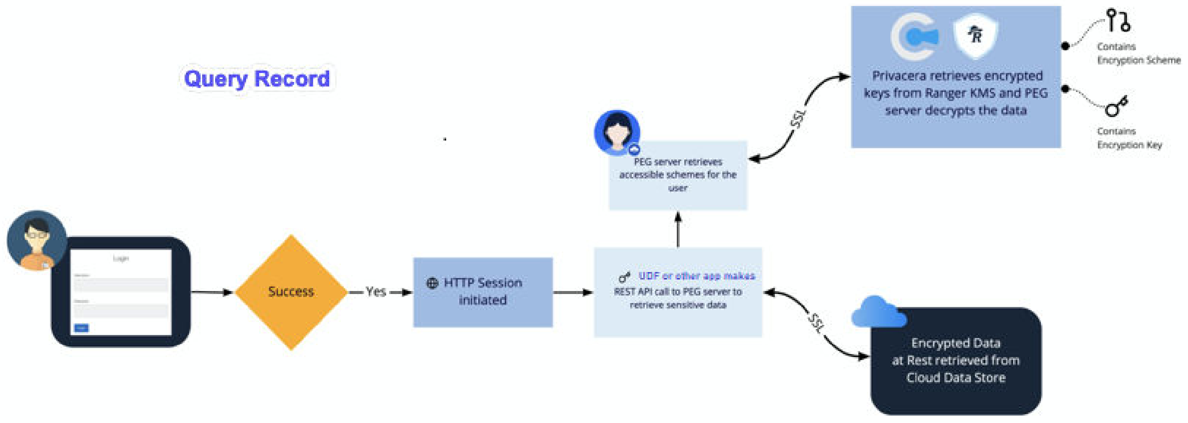


The Privacera encryption process is:

1. An endpoint is called to encrypt raw data.
   a. The scheme policy protecting access to encryption functions is checked.
   b. The encryption scheme encrypts the data according to its associated format, algorithm, and scope [32].
2. The data is encrypted.
3. An endpoint is called to decrypt the encrypted data.
   a. The scheme policy protecting access to encryption functions is checked.
   b. The same encryption scheme that encrypted the data is used to decrypt according to the encryption scheme's format, algorithm, and scope [32].
   c. The presentation scheme obfuscates the decrypted data for presentation to the user.

Encryption architecture and UDF flow

The following diagram shows the PEG architecture for viewing a record. For a description of the keys in this architecture, see Encryption keys [18].


Query Record
Login
Success
Yes
HTTP Session initiated
UDF or other app makes REST API call to PEG server to retrieve sensitive data
PEG server retrieves accessible schemes for the user
SSL
Privacera retrieves encrypted keys from Ranger KMS and PEG server decrypts the data
Contains Encryption Scheme
Contains Encryption Key
SSL
Encrypted Data at Rest retrieved from Cloud Data Store

1. A user queries sensitive data.
2. The Privacera Access Manager verifies the user access privileges to the data and the key (encryption scheme) used to decrypt the data.
3. If the user has access privileges to both the data and key, Privacera encryption requests Data Encryption Key (DEK) for the encryption scheme.
4. The Privacera Encryption Gateway (PEG) sends the Encrypted Data Encryption Key (EDEK) from the scheme to Ranger KMS to decrypt the DEK.
5. Ranger KMS authenticates the caller (the encryption module) and uses the KEK to decrypt EDEK and obtain the DEK.
6. The PEG obtains the DEK and decrypts the data.
7. The PEG returns the data to user.

Install Encryption on Privacera Platform

Privacera Encryption and the Privacera Encryption Gateway (PEG) are enabled in the Privacera Manager.

Use the instructions in the following links to install and enable the Privacera Manager components for encryption.

- Make sure your server instance meets the Encryption on Privacera Platform deployment specifications [8].
- Set up PEG and Cryptography with Ranger KMS on Privacera Platform [13].
- Enable self-signed certificates on Privacera Platform.
- Enable CA-signed certificates on Privacera Platform.
- Provide user access to Ranger KMS [14].

Encryption on Privacera Platform deployment specifications

These are the minimal specifications for running Privacera Encryption in production.

Hardware requirements

Hardware	Minimum Configuration
Number of CPUs	32
RAM	32 GB
Network bandwidth	10 Gbps

Software and server configuration

These are the specifications and configurations for Privacera Encryption software components.

Tomcat and Privacera Encryption settings

To configure Tomcat and Privacera Encryption, you must set properties in the following variable file:

- `~/privacera/privacera-manager/config/custom-vars/vars.peg.yml`

In the variable file, set the following properties:

```
PEG_SERVER_TOMCAT_MAX_THREADS: "1024"

PEG_SERVER_TOMCAT_CONNECTION_TIMEOUT: "20s"

PEG_SERVER_TOMCAT_MAX_CONNECTIONS : "1200"

PEG_SERVER_TOMCAT_MIN_SPARE_THREADS: "200"

PEG_SERVER_TOMCAT_ACCEPT_COUNT: "1000"

PRIVACERA_PEG_CONNECTIONPOOL: "500"
```

After setting these properties, you must update Privacera Manager.

Operational specifications

These specifications relate to the operational use of Privacera Encryption.

Batch together the elements in the PEG REST API endpoints in the `datalist` JSON array of the `/protect` or `/unprotect` request.

- **Minimum batch size**: 2,000 elements per request.
- **Maximum recommended batch size**: 15,000.
- **Maximum number of requests**: 1,800.

NOTE

Network latency can impact performance. Your network architecture should be optimized.

Configure Ranger KMS with Azure Key Vault on Privacera Platform

This topic shows how to configure Ranger Key Management Storage (KMS) system with Azure Key Vault to enable the use of data encryption. The master key for the encryption is created within the KMS and stored in Azure Key Vault. This section describes how to set up the connection from Ranger KMS to the Azure Key Vault to store the master key in the Azure key vault instead of the Ranger database.

NOTE

You can manually move the Ranger KMS from the Ranger database to the Azure Key Vault. For more information, refer to Migrate Ranger KMS master key on Privacera Platform [29].

Prerequisites

- If the authentication is done without SSL enabled, get the Key Vault URL, ClientId and Client Secret by following the steps in this topic, Connect to Azure Key Vault with Client ID and Client Secret on Privacera Platform [24].
- If the authentication is done with SSL enabled, get the Key Vault URL, ClientId and Certificate by following the steps in this topic, Connect to Azure Key Vault with a client ID and certificate on Privacera Platform [19].
- Configure Privacera Cryptography with Ranger KMS. For more information, refer to Set up PEG and Cryptography with Ranger KMS on Privacera Platform [13].

CLI configuration

1. SSH to the instance where Privacera is installed.
2. Run the following commands.

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.crypto.azurekeyvault.yml config/custom-vars/
   vi config/custom-vars/vars.crypto.azurekeyvault.yml
   ```
3. Edit the following properties. For property details and description, refer to the **Configuration Properties** below.

   ```
   AZURE_KEYVAULT_SSL_ENABLED: "<PLEASE_CHANGE>"
   AZURE_KEYVAULT_CLIENT_ID: "<PLEASE_CHANGE>"
   AZURE_KEYVAULT_CLIENT_SECRET: "<PLEASE_CHANGE>"
   AZURE_KEYVAULT_CERT_FILE: "<PLEASE_CHANGE>"
   AZURE_KEYVAULT_CERTIFICATE_PASSWORD: "<PLEASE_CHANGE>"
   AZURE_KEYVAULT_MASTERKEY_NAME: "<PLEASE_CHANGE>"
   AZURE_KEYVAULT_MASTER_KEY_TYPE: "<PLEASE_CHANGE>"
   AZURE_KEYVAULT_ZONE_KEY_ENCRYPTION_ALGO: "<PLEASE_CHANGE>"
   AZURE_KEYVAULT_URL: "<PLEASE_CHANGE>"
   ```
4. Run the following commands.

   ```
   cd ~/privacera/privacera-manager
   ./privacera-manager.sh update
   ```

Azure Key Vault properties on Privacera Platform

Property	Description	Example
AZURE_KEYVAULT_SSL_ENABLED	Activate Azure Key Vault.	true
AZURE_KEYVAULT_CLIENT_ID	Get the ID by following the **Pre-requisites** section above.	50fd7ca6-xxxx-xxxx-a13f-1xxxxxxxx
AZURE_KEYVAULT_CLIENT_SECRET	Get the client secret by following the **Pre-requisites** section above.	<AzureKeyVaultPassword>
AZURE_KEYVAULT_CERT_FILE	Get the file by following the **Pre-requisites** section above. Ensure the file is copied in the *config/ssl* folder, and give it a name.	azure-key-vault.pem
AZURE_KEYVAULT_CERTIFICATE_PASSWORD	Get the value by following the **Pre-requisites** section above.	certPass
AZURE_KEYVAULT_MASTERKEY_NAME	Enter the name of the master key. A key with this name will be created in Azure Key Vault.	RangerMasterKey
AZURE_KEYVAULT_MASTER_KEY_TYPE	Enter a type of master key. **Values**: RSA, RSA_HSM, EC, EC_HSM, OCT	RSA

Property	Description	Example
AZURE_KEY-VAULT_ZONE_KEY_ENCRYP-TION_ALGO	Enter an encryption algorithm for the master key. **Values**: RSA_OAEP, RSA_OAEP_256, RSA1_5, RSA_OAEP	RSA_OAEP
AZURE_KEYVAULT_URL	Get the URL by following the **Pre-requisites** section above.	https://key-vault.vault.azure.net/

Enable telemetry data collection on Privacera Platform

By default, the collection of telemetry data about the use of the Privacera Encryption Gateway (PEG) is disabled.

To enable telemetry data collection, follow these steps:

1. Copy the `vars.peg.yml` configuration file to `custom-vars/`:

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.peg.yml config/custom-vars/
   vi config/custom-vars/vars.peg.yml
   ```
2. Edit the following properties in `vars.peg.yml`. For a list of custom properties that can be configured for the Solr service, see Solr properties on Privacera Platform.

   ```
   PRIVACERA_PEG_SOLR_METRICS_ENABLE:"true"
   PRIVACERA_PEG_SOLR_URL:"<PLEASE_CHANGE>"
   PRIVACERA_PEG_SOLR_BASIC_AUTH_ENABLED:"<PLEASE_CHANGE>"
   PRIVACERA_PEG_SOLR_USER:"<PLEASE_CHANGE>"
   PRIVACERA_PEG_SOLR_PASSWORD:"<PLEASE_CHANGE>"
   PRIVACERA_PEG_SOLR_ZOOKEEPER_URL:"<PLEASE_CHANGE>"
   PRIVACERA_PEG_SOLR_COLLECTION:"<PLEASE_CHANGE>"
   PRIVACERA_PEG_SOLR_UPDATE_INTERVAL_SECONDS:"<PLEASE_CHANGE>"
   ```
3. Restart Privacera Manager:

   ```
   ./privacera-manager.sh update
   ```

   Telemetry data collection is enabled.

AWS S3 bucket encryption on Privacera Platform

You can set up server-side encryption for AWS S3 bucket to encrypt the resources in the bucket. Supported encryption types are Amazon S3 (SSE-S3), AWS Key Management Service (SSE-KMS), and Customer-Provided Keys (SSE-C). Encryption key is mandatory for the encryption type SSE-C and optional for SSE-KMS. No encryption key is required for SSE-S3. For more information, see Protecting data using server-side encryption in the AWS documentation.

Configure bucket encryption in dataserver

1. SSH to EC2 instance where Privacera Dataserver is installed.
2. Enable use of bucket encryption configuration in Privacera Dataserver:

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.dataserver.aws.yml config/custom-vars/
   vi config/custom-vars/vars.dataserver.aws.yml
   ```
3. Add the new property:

   ```
   DATA_SERVER_AWS_S3_ENCRYPTION_ENABLE:"true"DATA_SERVER_AWS_S3_ENCRYPTION_MAPPING:-"bu
   ```

Property	Description
DATA_SERV-ER_AWS_S3_ENCRYP-TION_ENABLE	Property to enable or disable the AWS S3 bucket encryption support.
DATA_SERV-ER_AWS_S3_ENCRYP-TION_MAPPING	Property to set the mapping of S3 buckets, encryption SSE type, and SSE key (base64 encoded). For example, "bucketC*,BucketD|SSE-KMS|<base64 encoded sse key>". The base64-encoded encryption key should be set for the following: 1) Encryption type is set to SSE-KMS and customer managed CMKs is used for encryption. 2) Encryption type is set to SSE-C.

Server-Side encryption with Amazon S3-Managed Keys (SSE-S3)

Supported S3 APIs for SSE-S3 Encryption:

- PUT Object
- PUT Object - Copy
- POST Object
- Initiate Multipart Upload

Bucket policy

```
{"Version":"2012-10-17","Id":"PutObjectPolicy","Statement":[{"Sid":"DenyIncorrectEncrypt
```

- Upload a test file.

```
aws s3 cp myfile.txt s3://{{sse-s3-encrypted-bucket}}/
```

Server-Side encryption with CMKs stored in AWS Key Management Service (SSE-KMS)

Supported APIs for SSE-KMS Encryption:

- PUT Object
- PUT Object - Copy
- POST Object
- Initiate Multipart Upload

Your IAM role should have kms:Decrypt permission when you upload or download an Amazon S3 object encrypted with an AWS KMS CMK. This is in addition to the kms:ReEncrypt, kms:GenerateData-Key, and kms:DescribeKey permissions.

AWS Managed CMKs (SSE-KMS)

Bucket Policy

```
{"Version":"2012-10-17","Id":"PutObjectPolicy","Statement":[{"Sid":"DenyIncorrectEncrypt
```

- Upload a test file.

```
aws s3 cp myfile.txt s3://{{sse-s3-encrypted-bucket}}/
```

Customer Managed CMKs (SSE-KMS)

Bucket Policy

```
{"Version":"2012-10-17","Id":"PutObjectPolicy","Statement":[{"Sid":"DenyIncorrectEncrypt
```

- Upload a test file.

```
aws s3 cp privacera_aws.sh s3://{{sse-kms-encrypted-bucket}}/
```

Server-Side encryption with Customer-Provided Keys (SSE-C)

Supported APIs for SSE-C Encryption:

- PUT Object
- PUT Object - Copy
- POST Object
- Initiate Multipart Upload
- Upload Part
- Upload Part - Copy
- Complete Multipart Upload
- Get Object
- Head Object
- Update the *privacera_aws_config.json* file with bucket and SSE-C encryption key.
  - Run AWS S3 upload.

    ```
    aws s3 cp myfile.txt s3://{{sse-c-encrypted-bucket}}/
    ```
  - Run head-object.

    ```
    aws s3api head-object --bucket {{sse-c-encrypted-bucket}} --key myfile.txt
    ```

Sample keys:

Key	Value
AES256-bit key	`E1AC89EFB167B29ECC15FF75CC5C2C3A`
Base64-encoded encryption key (sseKey)	`echo -n "E1AC89EFB167B29ECC15FF75CC5C2C3A" | openssl enc -base64`
Base64-encoded 128-bit MD5 digest of the encryption key	`echo -n "E1AC89EFB167B29ECC15FF75CC5C2C3A" | openssl dgst -md5 -binary | openssl enc -base64`

Set up PEG and Cryptography with Ranger KMS on Privacera Platform

This topic covers how to set up and use Privacera Cryptography and Privacera Encryption Gateway (PEG) using Ranger KMS.

1. SSH to the instance where Privacera is installed.
2. Create a crypto configuration file, and set the value of the Ranger KMS Master Key Password.

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.crypto.yml config/custom-vars/
   vi config/custom-vars/vars.crypto.yml
   ```
3. Assign a password to the `RANGER_KMS_MASTER_KEY_PASSWORD` such as "Str0ngP@ssw0rd".

   ```
   RANGER_KMS_MASTER_KEY_PASSWORD: "<PLEASE_CHANGE>"
   ```
4. Run the following command.

   ```
   cp config/sample-vars/vars.peg.yml config/custom-vars/
   ```
5. (Optional) If you want to customize PEG configuration further, you can add custom PEG properties. For more information, refer to PEG custom properties [14].

   For example, by default, the username and password for the PEG service is padmin/padmin. If you choose to change it, refer to Add custom properties using Privacera Manager on Privacera Platform.
6. Run Privacera Manager to update the Privacera Platform configuration:

   ```
   cd ~/privacera/privacera-manager
   ./privacera-manager.sh update
   ```
7. If this is a Kubernetes deployment, update all Privacera services:

```
./privacera-manager.sh update
```

Provide user access to Ranger KMS

To provide user access to the keys needed for encryption, you must create a policy in Apache Ranger KMS. To do so:

1. Log in to the Ranger portal and select **Access Managemer** > **Resource Based Policies**.
2. In the KMS section, click **privacera_kms**.
3. In the List of Policies: privacera_kms section, click **Add New Policy**.
4. In the Create Policy screen, enter the following information to create a policy and provide access to the user:
   - **Policy Name**: Enter the access policy name.
   - **Policy Label**: Enter a label name (optional).
   - **Key Name**: Type a character to list the existing key names that are already generated in Ranger.
   - **Description**: Enter a description for the policy.
   - **Audit Logging**: Toggle Yes or No.
5. In the **Allow Conditions** section, select the following:
   - **SelectRole:** Enter or select from existing roles.
   - **Select Group**: Enter or select from existing group.
   - **SelectUser**: This is the username that will be used in the encryption API - select or enter a new user name.
   - **Add Permissions:** Select user permissions - Create, Delete, Rollover, Set Key Material, Get, Get Keys, Get Metadata, Generate EEK, Decrypt EEK, Select/Deselect All.
   - **Delegate Admin**: If this user is delegate as the admin.
6. Similarly, for specific users, you can select users to **Exclude from Allow Conditions**, **Deny Conditions**, **Exclude from Deny Conditions**.
7. Click **Add** to save the policy.

Provide user access for Encryption Service

To set user access for the Encryption Service in the Apache Ranger KMS, follow these steps:

1. Log in to the Ranger portal.
2. In the **Access Manager** tab, select `privacera_kms` policy.
3. Click the edit button next to the **all - key** policy.
4. In the **Allow Conditions** section, search and select `privacera_service_discovery` user from the **Select User** dropdown menu.

PEG custom properties

The following table contains the list of custom properties that can be configured for PEG. To use a custom property from the table, add it to the `vars.peg.yml` YML file in the `custom-vars` folder:

Property	Description	Values	Default Value
`PEG_IMAGE_NAME`			
`PEG_IMAGE_TAG`			
`USERSYNC_IMAGE_NAME`			
`PEG_ENABLE`			
`PEG_SSL_ENABLE`			
`PEG_SSL_SELF_SIGNED`			
`USERSYNC_RANGER_URL`			
`PEG_INTERNAL_PORT`			
`PEG_PORT`	Property to change the default port number for PEG.		`6869`
`PEG_PROTOCOL`			

Property	Description	Values	Default Value
PEG_PROTOCOL_URL			
USERSYNC_SYNC_LDAP_USER_SEARCH_BASE			
PEG_SERVICE_NAME			
USERSYNC_SYNC_LDAP_OBJECT_CLASS			
PEG_HOST_NAME			
USERSYNC_SYNC_LDAP_USER_EMAIL_ADDRESS_ATTRIBUTE			
PEG_SVC_IP			
PEG_EXTERNAL_HOST			
USERSYNC_SYNC_LDAP_SSL_ENABLED			
PEG_URL			
USERSYNC_SYNC_LDAP_SSL_TRUSTSTORE_FILE			
PEG_EXTERNAL_URL			
USERSYNC_SYNC_LDAP_SSL_TRUSTSTORE_PASSWORD			
PEG_URL_IP			
PEG_PORTAL_USERNAME	Username used by PEG to access Privacera Portal.		padmin
PEG_PORTAL_PASSWORD	Password used by PEG to access Privacera Portal.		{{PORTAL_PADMIN_PASSWORD}}
PEG_USERNAME	Username of PEG API credentials to access the PEG API services.		padmin
PEG_PASSWORD	Password of PEG API credentials to access the PEG API services.		
PEG_LOG4J_LEVEL			
PEG_TOMCAT_BASE_DIR			
PEG_SSL_KEY_STORE			
PEG_SSL_TRUST_STORE			
PEG_KEYSTORE_PASSWORD			
PEG_TRUSTSTORE_PASSWORD			
PEG_KEYSTORE_ALIAS			
PEG_SSL_KEYSTORETYPE			
USERSYNC_SYNC_GROUP_OBJECT_CLASS			
PEG_PORTAL_AUTH			
PEG_METRICS_ENABLE			
PEG_METRICS_ENABLE_GRAPHITE			
PEG_METRICS_ENABLE_JVM			
USERSYNC_SYNC_PAGED_RESULTS_SIZE			
PEG_INMEM_AUTH			
PEG_SSL_SIGNED_PEM_FULL_CHAIN			
PEG_SSL_SIGNED_PEM_PRIVATE_KEY			
PEG_SSL_PKCS12_PASSWORD			
PEG_SSL_SIGNED_CERT_FORMAT			
PEG_SSL_SIGNED_PKCS12_ALIAS			
PEG_SSL_SIGNED_PKCS12_FILE			
PEG_AUTHORIZATION_ENABLED			
PEG_AUTHORIZER_IMPL			
USERSYNC_KERBEROS_KEYTAB			
PEG_ENCRYPT_SECRETS			
PEG_SECURE_JCEKS_FILE_PATHS			
PEG_SECURE_JCEKS_KEYS			

Property	Description	Values	Default Value
PEG_SECURE_JCEKS_KEYPREFIX			
PEG_ENCRYPT_PROPS_LIST			
PEG_K8S_PVC_NAME			
PEG_K8S_PVC_STORAGE_SIZE_MB			
PEG_K8S_PVC_STORAGE_SIZE			
PEG_K8S_STORAGE_PROVISIONER			
PEG_K8S_SC_NAME			
PEG_K8S_PV_ENCRYPTED			
PEG_K8S_PV_KEY			
USERSYNC_AZUREAD_PASSWORD			
PEG_REPLICAS_MIN			
PEG_REPLICAS_MAX			
PEG_K8S_LOADBALANCER_EXTERNAL			
PEG_K8S_ANNOTATION_LOADBALANCER_ANNOTATION			
PEG_K8S_MEM_LIMITS			
PEG_K8S_MEM_REQUESTS			
PEG_K8S_CPU_LIMITS			
PEG_K8S_CPU_REQUESTS			
SYNC_AZUREAD_USER_SERVICE_PRINCIPAL_ENABLED			
SYNC_AZUREAD_USER_SERVICE_PRINCIPAL_USERNAME_RETRIVAL_FROM			
USERSYNC_RANGER_USERSYNC_COOKIE			
USERSYNC_LOGDIR			
USERSYNC_ENCRYPT_SECRETS			
USERSYNC_SECRETS_FILE			
USERSYNC_SECRETS_KEYSTORE_PASSWORD			
USERSYNC_ENCRYPT_PROPS_LIST			
USERSYNC_AUTH_ADD_ETCHOST			
USERSYNC_AUTH_IP			
USERSYNC_AUTH_HOST			
USERSYNC_K8S_MEM_LIMITS			
USERSYNC_K8S_MEM_REQUESTS			
USERSYNC_K8S_CPU_LIMITS			
USERSYNC_K8S_CPU_REQUESTS			
USERSYNC_PASSWORDS_LIST			
Memory Variables			
PEG_HEAP_MIN_MEMORY_MB	Minimum Java Heap memory in MB used by PEG. For example, PEG_HEAP_MIN_MEMORY_MB: "1024"		
PEG_HEAP_MIN_MEMORY	Minimum Java Heap memory used by PEG. Setting this value will override PEG_HEAP_MIN_MEMORY_MB. For example, PEG_HEAP_MIN_MEMORY: "1g"		
PEG_HEAP_MAX_MEMORY_MB	Maximum Java Heap memory in MB used by PEG. For example, PEG_HEAP_MAX_MEMORY_MB: "1024"		
PEG_HEAP_MAX_MEMORY	Maximum Java Heap memory used by PEG. Setting this value will override PEG_HEAP_MAX_MEMORY_MB. For example, PEG_HEAP_MAX_MEMORY: "1g"		

Property	Description	Values	Default Value
`PEG_K8S_MEM_REQUESTS_MB`	Minimum amount of Kubernetes memory in MB to be requested by PEG. For example, `PEG_K8S_MEM_REQUESTS_MB: "1024"`		
`PEG_K8S_MEM_REQUESTS`	Minimum amount of Kubernetes memory to be used by PEG. Setting this value will override `PEG_K8S_MEM_REQUESTS_MB`. For example, `PEG_K8S_MEM_REQUESTS: "1G"`		
`PEG_K8S_MEM_LIMITS_MB`	Maximum amount of Kubernetes memory in MB to be requested by PEG. For example, `PEG_K8S_MEM_LIMITS_MB: "1024"`		
`PEG_K8S_MEM_LIMITS`	Maximum amount of Kubernetes memory to be used by PEG. Setting this value will override `PEG_K8S_MEM_LIMITS_MB`. For example, `PEG_K8S_MEM_LIMITS: "1G"`		
`PEG_CPU_MIN`	Minimum amount of Kubernetes CPU to be requested by PEG. For example, `PEG_CPU_MIN: "0.5"`		
`PEG_CPU_MAX`	Maximum amount of Kubernetes CPU to be used by PEG. For example, `PEG_CPU_MAX: "0.5"`		

Enable Encryption on PrivaceraCloud

PrivaceraCloud Privacera Encryption Gateway (PEG) supports two API REST methods: protect and unprotect. It uses Basic Auth (Base64 encoding) authenticated against a single configured service user.

To enable encryption for your applications:

1. On the **Account** page, click the **ENABLE** button next to the Privacera Encryption.
   The **Privacera Encryption Configuration** page appears.
2. In the **BASIC** tab, enter the following information:
   - Enter credentials (`Username` and `Password`) for a PEG service user. These are the Basic Authentication values for the PEG API requests.
   - Enter a value for a `secret`. This value will be used as a shared secret when configuring embedded encryption using the Privacera Crypto Jar, for use in Databricks. See Enable Privacera Encryption services in Databricks SQL on PrivaceraCloud for additional setup details, if using PEG with Databricks SQL and User-Defined Functions (UDFs).
3. In the **ADVANCED** tab, you can add custom properties.
4. Using the **IMPORT PROPERTIES** and **EXPORT PROPERTIES** button, you can browse and import/export properties.
5. Click **SAVE**.

Thereafter, use the toggle to either disable or enable encryption and use the pen icon to modify the configuration.

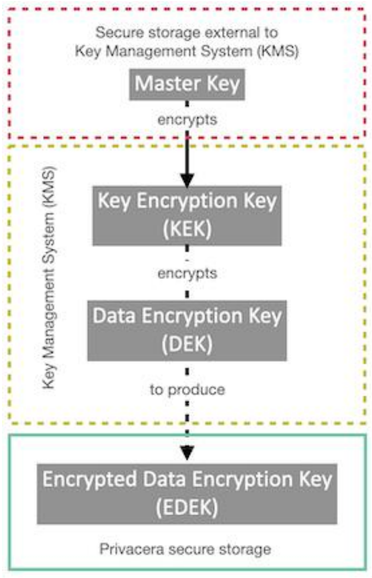
NOTE

When you update the encryption configuration, you should restart it. This is to ensure that your updated configuration functions properly.

Encryption keys

Key management is a critical part of preventing the compromise of your encryption keys for both data-at-rest and data-in-transit. Encryption keys must be secured by storing them in a separate Key Management System (KMS). Privacera uses Apache Ranger KMS, where keys are stored in an encrypted format.

Privacera Encryption uses the following types of encryption keys:


Secure storage external to Key Management System (KMS)
Master Key
encrypts
Key Management System (KMS)
Key Encryption Key (KEK)
encrypts
Data Encryption Key (DEK)
to produce
Encrypted Data Encryption Key (EDEK)
Privacera secure storage

Master Key

The Master Key encrypts the Key Encryption Keys (KEK).

The Master Key is stored outside of the KMS database or externally on a hardware security module (HSM).

Key Encryption Key (KEK)

A KEK encrypts the Data Encryption Key (DEK). The Master Key encrypts KEKs.

KEKs are stored in Apache Ranger KMS. Apache Ranger KMS uses the KEKs to:

- Encrypt DEKs to create Encrypted Data Encryption Keys (EDEKs)
- Decrypt EDEKs

Manage Key Encryption Keys (KEKs) on Privacera Platform

If you delete a KEK, all of the associated encrypted data cannot be decrypted.

KEKs should be rolled over at regular intervals, such as every 12 months. You can increase the frequency depending on how extensively the KEK is used. For more information, see Rollover encryption keys on Privacera Platform [19].

Data Encryption Key (DEK)

The Data Encryption Key (DEK) encrypts and decrypts your data.

Each encryption scheme created in the Privacera Portal is mapped to a unique DEK. The user must have key access privileges by way of a scheme policy to encrypt or decrypt data with the DEK.

The DEK is stored in an encrypted format as an Encrypted Data Encryption Key (EDEK). The key used to encrypt the DEK is managed by Apache Ranger KMS.

Encrypted Data Encryption Key (EDEK)

The EDEK is the encrypted DEK and is encrypted with a KEK. A KEK is required to decrypt an EDEK. EDEKs are stored and managed by Privacera.

Rollover encryption keys on Privacera Platform

Privacera uses Apache Ranger to encrypt data. You can rollover encryption keys from the Apache Ranger UI or using the REST API `/keys/key`. If a key was used to encrypt several terabytes of data, it would be computationally intensive and time-consuming to rollover the keys. During the key rollover process, which first decrypts the data using existing keys and then re-encrypts the data using the new keys, your data is not available.

To overcome this challenge, Privacera encrypts Data Encryption Keys (DEKs) that are used to encrypt the data. A separate set of keys called Key Encryption Keys (KEKs) are used to encrypt the DEKs. The term “rollover” means rotating the KEKs instead of the DEKs. Even if you have ten thousand keys, the process to rotate the KEKs can be completed extremely quickly.

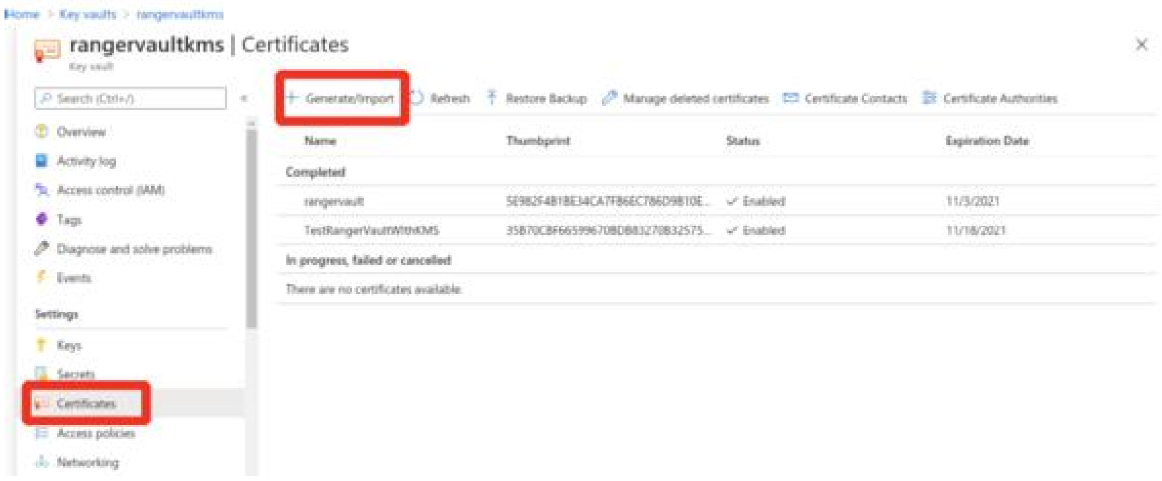
To rollover encryption keys using Apache Ranger:

1. Log in to Ranger at `https://<your_privacera_hostname>:6080` using “keyadmin” credentials.
2. Hover your cursor over the Encryption menu item and select **Key Manager**.
3. From the **Select Service** dropdown menu, select **privacera_kms**.
   The current key entries are displayed.
4. Click the pencil icon for the key you want to rollover.
5. Click **OK rollover**.
   The Ranger rollover Key API is called, which decrypts the DEKs that were encrypted using the previous key, creates a new key, and encrypts the DEKs using the newly generated key.

Connect to Azure Key Vault with a client ID and certificate on Privacera Platform

To configure a connection to the Azure Key Vault with ID and Certificate:

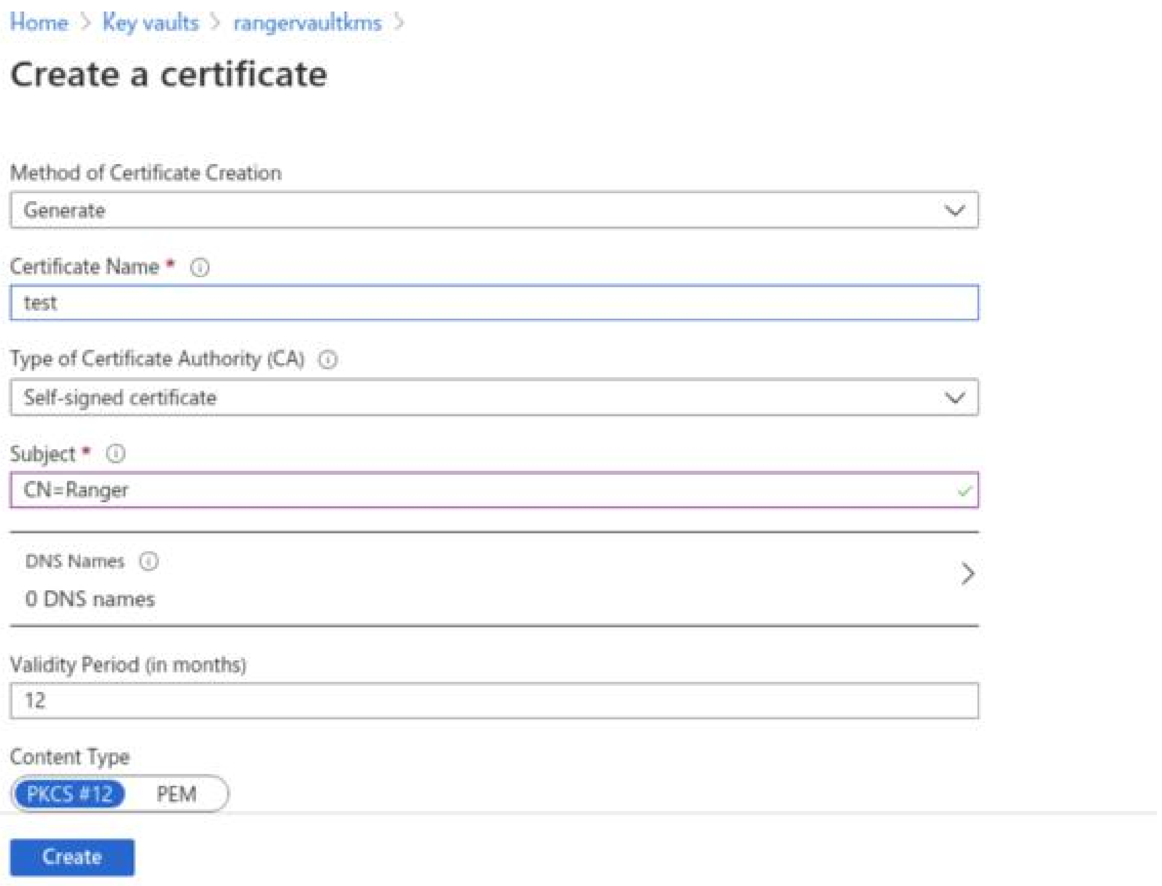
1. Follow the same steps as in **Generate the Client ID** in the topic Connect to Azure Key Vault with Client ID and Client Secret on Privacera Platform [24].
2. Go to the Key Vault generated and select the **Certificates>Generate/Import**.



Name	Thumbprint	Status	Expiration Date
Completed			
rangervault	5E982F4B1BE34CA7FB6EC786D9B10E...	✓ Enabled	11/3/2021
TestRangerVaultWithKMS	35B70CBF66599670BDB83270B32575...	✓ Enabled	11/18/2021
In progress, failed or cancelled			
There are no certificates available.			

You have the option to generate a certificate outside the vault and import it here.

3. Select **Generate** to generate a certificate.
4. Enter the certificate details as shown below:



Home > Key vaults > rangervaultkms >

Create a certificate

Method of Certificate Creation: Generate

Certificate Name *: test

Type of Certificate Authority (CA): Self-signed certificate

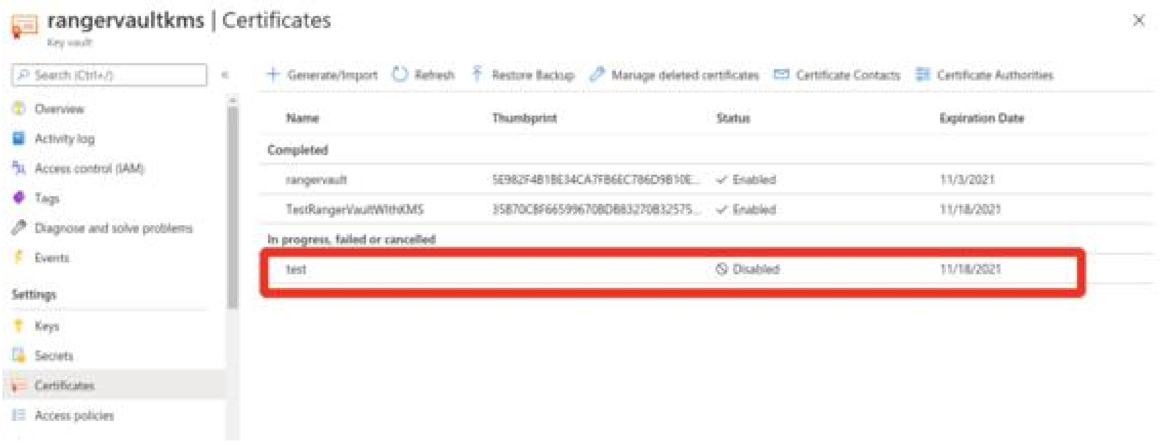
Subject *: CN=Ranger

DNS Names: 0 DNS names

Validity Period (in months): 12

Content Type: [✓ PKCS #12] PEM

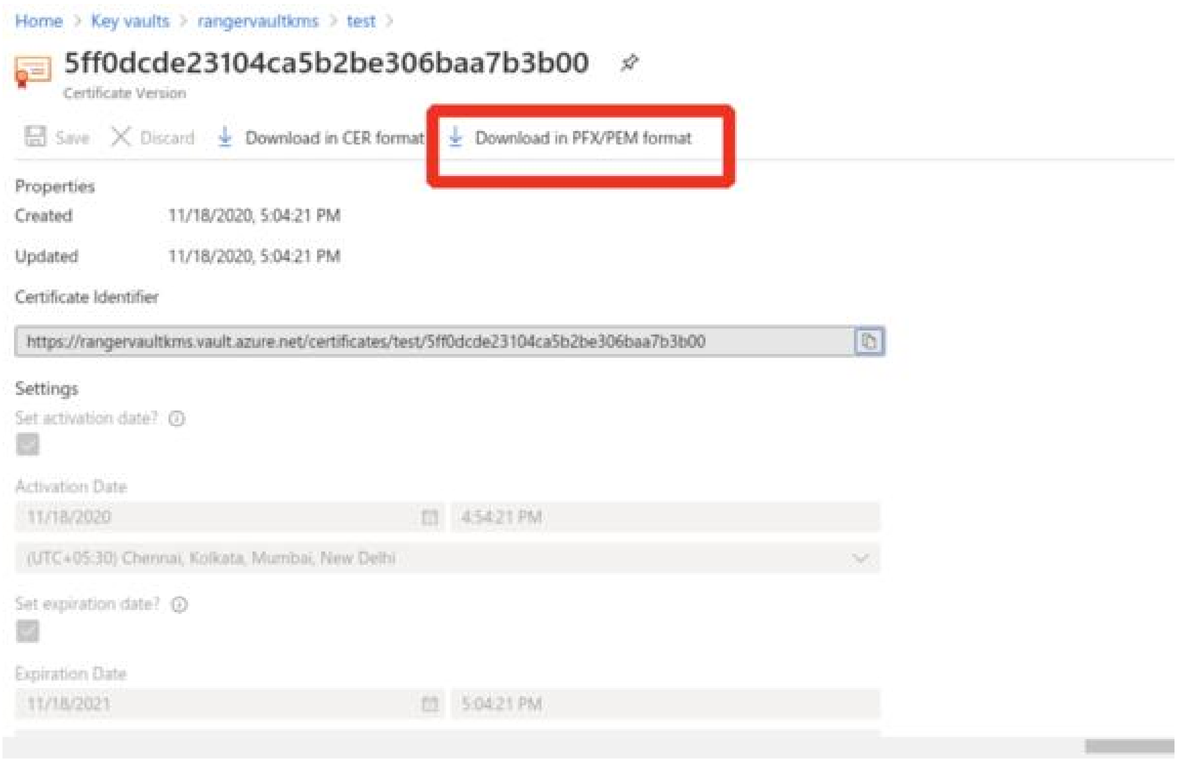
5. In the example shown, a certificate '**test**' is generated.



rangervaultkms | Certificates
Key vault

Name	Thumbprint	Status	Expiration Date
Completed			
rangervault	5E982F4B1BE34CA7FB6EC786D9B10E...	✓ Enabled	11/3/2021
TestRangerVaultWithKMS	35B70CBF665996708DB83270B32575...	✓ Enabled	11/18/2021
In progress, failed or cancelled			
test		Disabled	11/18/2021

6. Click on the certificate that is disabled and enable it.
7. Click **open the certificate** and download it.


Home > Key vaults > rangervaultkms > test >
5ff0dcde23104ca5b2be306baa7b3b00
Certificate Version
Save
Discard
Download in CER format
Download in PFX/PEM format

Properties

Created 11/18/2020, 5:04:21 PM

Updated 11/18/2020, 5:04:21 PM

Certificate Identifier

https://rangervaultkms.vault.azure.net/certificates/test/5ff0dcde23104ca5b2be306baa7b3b00

Settings

Set activation date? ✓

Activation Date

11/18/2020 4:54:21 PM

(UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi

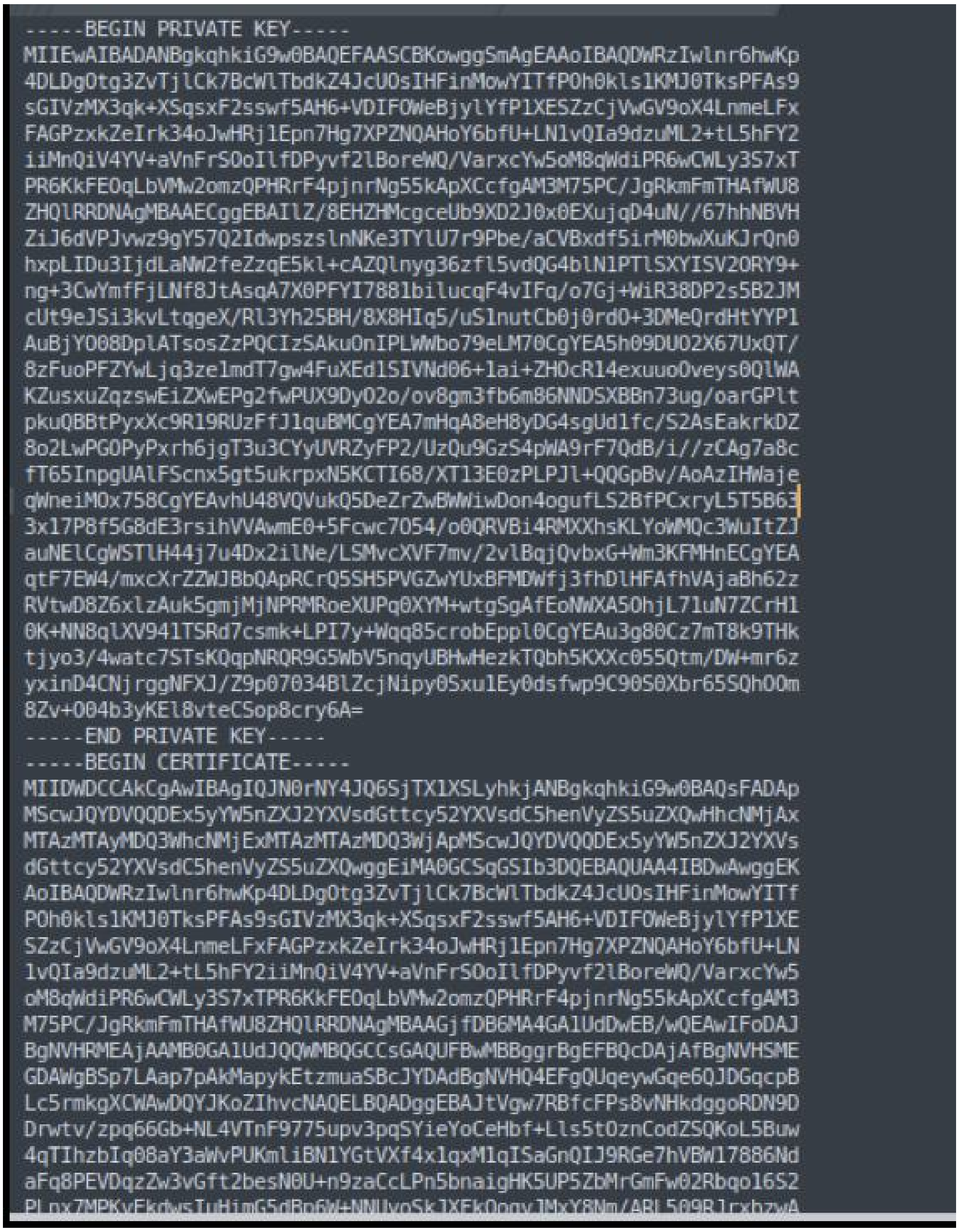
Set expiration date? ✓

Expiration Date

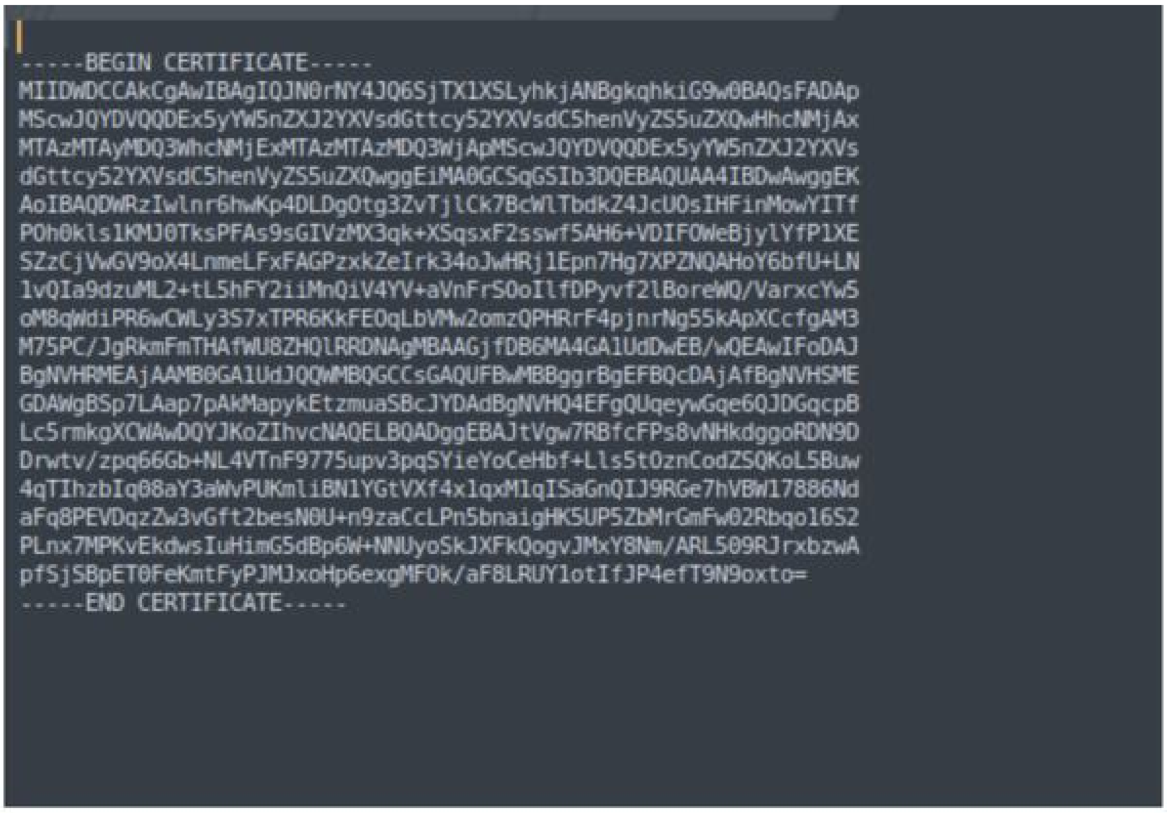
11/18/2021 5:04:21 PM

Download the certificate and. copy the certificate to the SSL folder: ~/privacera/privacera-manag-er/config/ssl/.

8. Open the certificate and delete the private key and save the public certificate as shown in this example:

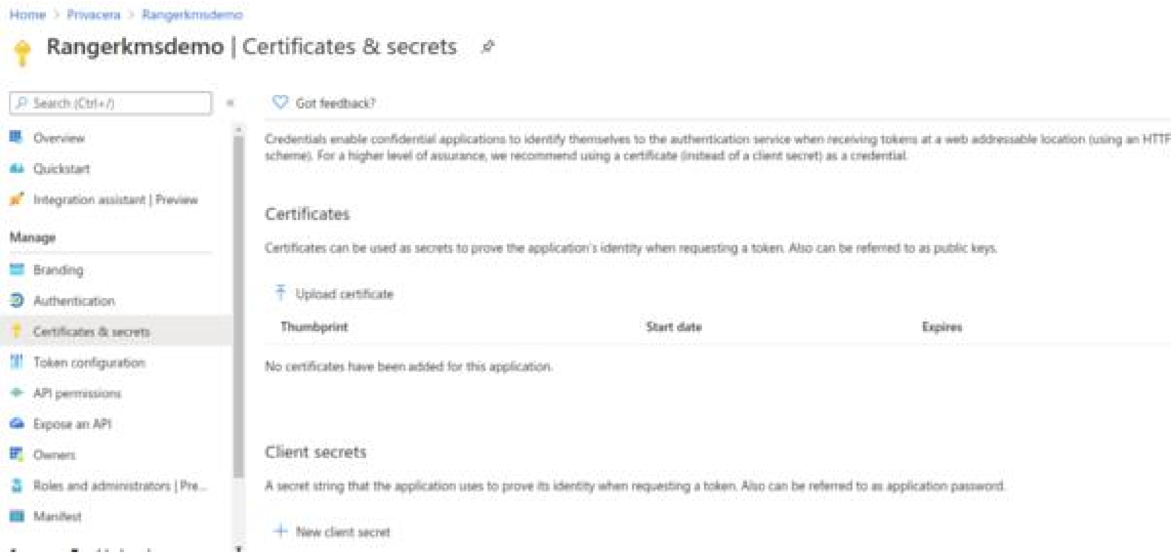


```
-----BEGIN PRIVATE KEY-----
MIIEwAIBADANBgkqhkiG9w0BAQEFAASCBKowggSmAgEAAoIBAQDWRzIwlnr6hwKp
4DLDgOtg3ZvTjlCk7BcWlTbdkZ4JcUOsIHFinMowYITfPOh0kls1KMJ0TksPFAs9
sGIVzMX3qk+XSqsxF2sswf5AH6+VDIFOWeBjylYfP1XESZzCjVwGV9oX4LnmeLFx
FAGPzxkZeIrk34oJwHRj1Epn7Hg7XPZNQAHoY6bfU+LN1vQIa9dzuML2+tL5hFY2
iiMnQiV4YV+aVnFrSOoIlfDPyvf2lBoreWQ/VarxcYw5oM8qWdiPR6wCWLy3S7xT
PR6KkFEOqLbVMw2omzQPHRrF4pjnrNg55kApXCcfgAM3M75PC/JgRkmFmTHAfWU8
ZHQlRRDNAgMBAAECggEBAIlZ/8EHZHMcgceUb9XD2J0x0EXujqD4uN//67hhNBVH
ZiJ6dVPJvwz9gY57Q2IdwpszslnNKe3TYlU7r9Pbe/aCVBxdf5irM0bwXuKJrQn0
hxpLIDu3IjdLaNW2feZzqE5kl+cAZQlnyg36zfl5vdQG4blN1PTlSXYISV2ORY9+
ng+3CwYmfFjLNf8JtAsqA7X0PFYI7881bilucqF4vIFq/o7Gj+WiR38DP2s5B2JM
cUt9eJSi3kvLtqgeX/Rl3Yh25BH/8X8HIq5/uS1nutCb0j0rdO+3DMeQrdHtYYP1
AuBjYO08DplATsosZzPQCIzSAkuOnIPLWWbo79eLM70CgYEA5h09DUO2X67UxQT/
8zFuoPFZYwLjq3ze1mdT7gw4FuXEd1SIVNd06+1ai+ZHOcR14exuuoOveys0QlWA
KZusxuZqzswEiZXwEPg2fwPUX9DyO2o/ov8gm3fb6m86NNDSXBBn73ug/oarGPlt
pkuQBBtPyxXc9R19RUzFfJ1quBMCgYEA7mHqA8eH8yDG4sgUd1fc/S2AsEakrkDZ
8o2LwPGOPyPxrh6jgT3u3CYyUVRZyFP2/UzQu9GzS4pWA9rF7QdB/i//zCAg7a8c
fT65InpgUAlFScnx5gt5ukrpxN5KCTI68/XT13E0zPLPJl+QQGpBv/AoAzIHWaje
qWneiMOx758CgYEAvhU48VQVukQ5DeZrZwBWWiwDon4ogufLS2BfPCxryL5T5B63
3x17P8f5G8dE3rsihVVAwmE0+5Fcwc7054/o0QRVBi4RMXXhsKLYoWMQc3WuItZJ
auNElCgWSTlH44j7u4Dx2ilNe/LSMvcXVF7mv/2vlBqjQvbxG+Wm3KFMHnECgYEA
qtF7EW4/mxcXrZZWJBbQApRCrQ5SH5PVGZwYUxBFMDWfj3fhDlHFAfhVAjaBh62z
RVtwD8Z6xlzAuk5gmjMjNPRMRoeXUPq0XYM+wtgSgAfEoNWXA5OhjL71uN7ZCrH1
0K+NN8qlXV941TSRd7csmk+LPI7y+Wqq85crobEppl0CgYEAu3g80Cz7mT8k9THk
tjyo3/4watc7STsKQqpNRQR9G5WbV5nqyUBHwHezkTQbh5KXXc055Qtm/DW+mr6z
yxinD4CNjrggNFXJ/Z9p07034BlZcjNipy05xu1Ey0dsfwp9C9050Xbr65SQhOOm
8Zv+O04b3yKEl8vteCSop8cry6A=
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIDWDCCAkCgAwIBAgIQJN0rNY4JQ6SjTX1XSLyhkjANBgkqhkiG9w0BAQsFADAp
MScwJQYDVQQDEx5yYW5nZXJ2YXVsdGttcy52YXVsdC5henVyZS5uZXQwHhcNMjAx
MTAzMTAyMDQ3WhcNMjExMTAzMTAzMDQ3WjApMScwJQYDVQQDEx5yYW5nZXJ2YXVs
dGttcy52YXVsdC5henVyZS5uZXQwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQDWRzIwlnr6hwKp4DLDgOtg3ZvTjlCk7BcWlTbdkZ4JcUOsIHFinMowYITf
POh0kls1KMJ0TksPFAs9sGIVzMX3qk+XSqsxF2sswf5AH6+VDIFOWeBjylYfP1XE
SZzCjVwGV9oX4LnmeLFxFAGPzxkZeIrk34oJwHRj1Epn7Hg7XPZNQAHoY6bfU+LN
1vQIa9dzuML2+tL5hFY2iiMnQiV4YV+aVnFrSOoIlfDPyvf2lBoreWQ/VarxcYw5
oM8qWdiPR6wCWLy3S7xTPR6KkFEOqLbVMw2omzQPHRrF4pjnrNg55kApXCcfgAM3
M75PC/JgRkmFmTHAfWU8ZHQlRRDNAgMBAAGjfDB6MA4GA1UdDwEB/wQEAwIFoDAJ
BgNVHRMEAjAAMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjAfBgNVHSME
GDAWgBSp7LAap7pAkMapykEtzmuaSBcJYDAdBgNVHQ4EFgQUqeywGqe6QJDGqcpB
Lc5rmkgXCWAwDQYJKoZIhvcNAQELBQADggEBAJtVgw7RBfcFPs8vNHkdggoRDN9D
Drwtv/zpq66Gb+NL4VTnF9775upv3pqSYieYoCeHbf+Lls5tOznCodZSQKoL5Buw
4qTIhzbIq08aY3aWvPUKmliBN1YGtVXf4x1qxM1qISaGnQIJ9RGe7hVBW17886Nd
aFq8PEVDqzZw3vGft2besN0U+n9zaCcLPn5bnaigHK5UP5ZbMrGmFw02Rbqo16S2
PLpx7MPKvFkdwsIuHimG5dBo6W+NNUvoSkJXFkOqqvJMxY8Nm/ABL509BJrxbzwA
```



```
-----BEGIN CERTIFICATE-----
MIIDWDCCAkCgAwIBAgIQJN0rNY4JQ6SjTX1XSLyhkjANBgkqhkiG9w0BAQsFADAp
MScwJQYDVQQDEx5yYW5nZXJ2YXVsdGttcy52YXVsdC5henVyZS5uZXQwHhcNMjAx
MTAzMTAyMDQ3WhcNMjExMTAzMTAzMDQ3WjApMScwJQYDVQQDEx5yYW5nZXJ2YXVs
dGttcy52YXVsdC5henVyZS5uZXQwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQDWRzIwlnr6hwKp4DLDg0tg3ZvTjlCk7BcWlTbdkZ4JcU0sIHFinMowYITf
POh0kls1KMJ0TksPFAs9sGIVzMX3qk+XSqsxF2sswf5AH6+VDIFOWeBjylYfP1XE
SZzCjVwGV9oX4LnmeLFxFAGPzxkZeIrk34oJwHRj1Epn7Hg7XPZNQAHoY6bfU+LN
1vQIa9dzuML2+tL5hFY2iiMnQiV4YV+aVnFrSOoIlfDPyvf2lBoreWQ/VarxcYw5
oM8qWdiPR6wCWLy3S7xTPR6KkFEOqLbVMw2omzQPHRrF4pjnrNg55kApXCcfgAM3
M75PC/JgRkmFmTHAfWU8ZHQlRRDNAgMBAAGjfDB6MA4GA1UdDwEB/wQEAwIFoDAJ
BgNVHRMEAjAAMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjAfBgNVHSME
GDAWgBSp7LAap7pAkMapykEtzmuaSBcJYDAdBgNVHQ4EFgQUqeywGqe6QJDGqcpB
Lc5rmkgXCWAwDQYJKoZIhvcNAQELBQADggEBAJtVgw7RBfcFPs8vNHkdggoRDN9D
Drwtv/zpq66Gb+NL4VTnF9775upv3pqSYieYoCeHbf+Lls5tOznCodZSQKoL5Buw
4qTIhzbIq08aY3aWvPUKmliBN1YGtVXf4x1qxM1qISaGnQIJ9RGe7hVBW17886Nd
aFq8PEVDqzZw3vGft2besN0U+n9zaCcLPn5bnaigHK5UP5ZbMrGmFw02Rbqo16S2
PLnx7MPKvEkdwsIuHimG5dBp6W+NNUyoSkJXFkQogvJMxY8Nm/ARL509RJrxbzwA
pfSjSBpET0FeKmtFyPJMJxoHp6exgMFOk/aF8LRUY1otIfJP4efT9N9oxto=
-----END CERTIFICATE-----
```

9. Upload the certificate to the Azure application that was created as follows:



Home > Privacera > Rangerkmsdemo

Rangerkmsdemo | Certificates & secrets

Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTF scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Certificates

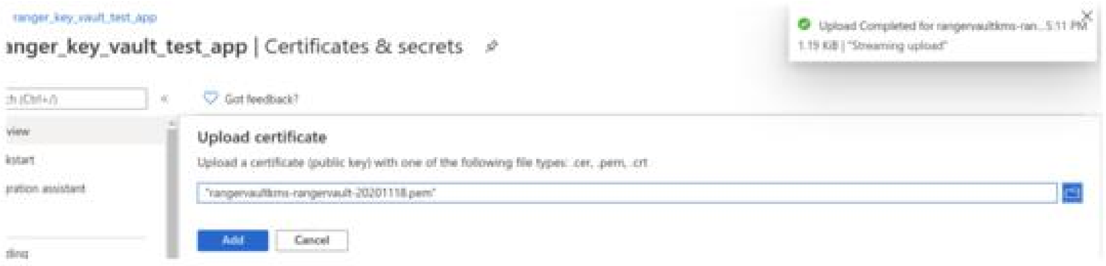
Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

Thumbprint	Start date	Expires

No certificates have been added for this application.

Client secrets

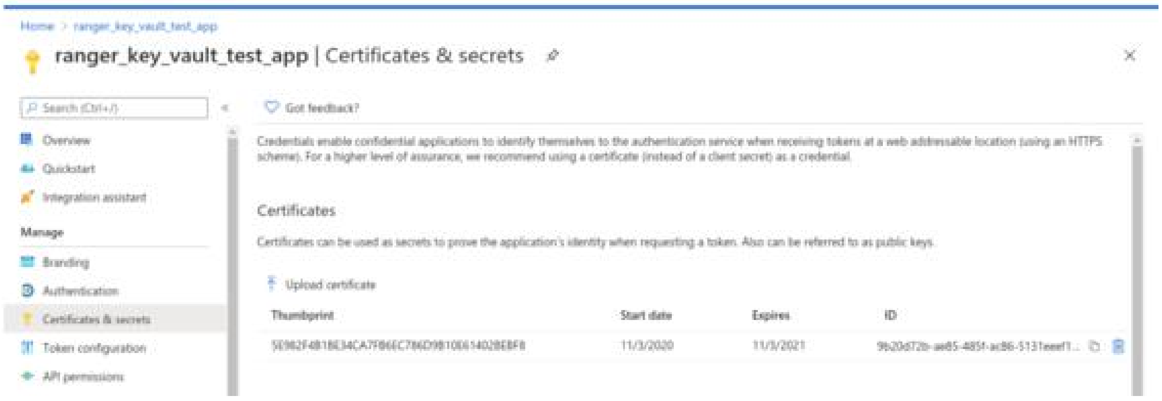
A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.


ranger_key_vault_test_app
anger_key_vault_test_app | Certificates & secrets
Got feedback?
Upload certificate
Add
Cancel

Upload Completed for rangervaultkms-ran... 5:11 PM
1.19 KiB | "Streaming upload"

Upload a certificate (public key) with one of the following file types: .cer, .pem, .crt

"rangervaultkms-rangervault-20201118.pem"



Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

Thumbprint	Start date	Expires	ID
5E982F4B1BE34CA7FB6EC786D9B10E61402BEBF8	11/3/2020	11/3/2021	9b20d72b-ae85-485f-ac86-5131eeef1...

10. Go the Key vault that was created and click on **Access Policies**.
11. Follow the instructions in Add the Access Policy [27].

NOTE
The certificate path should be as it is show in the ranger/kms/install.properties and cannot change. Also, if you need a password for the certificate, add it in the .properties file. All fields in the .properties file are required and cannot be removed. Value can be null/dummy.

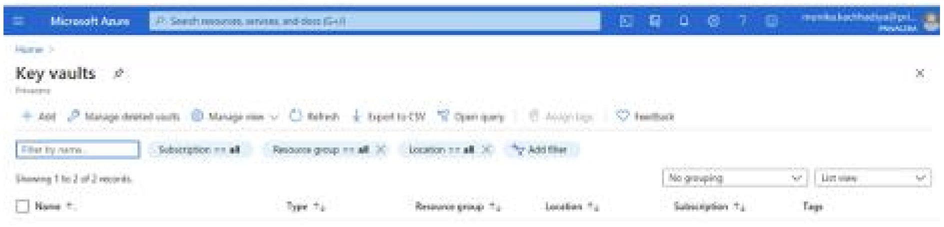
Connect to Azure Key Vault with Client ID and Client Secret on Privacera Platform

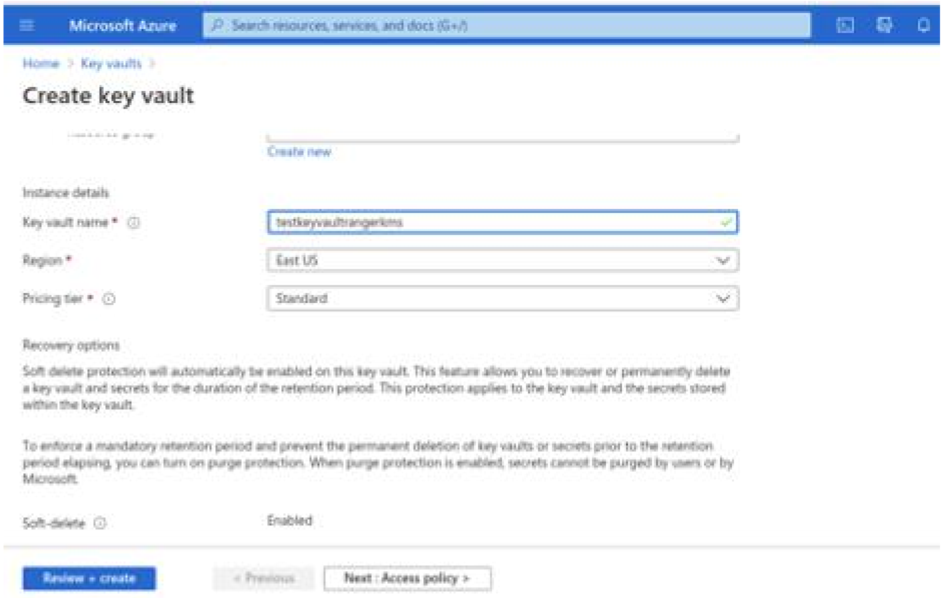
To configure a connection to the Azure Key Vault with ID and Secret:

1. Generate the Client ID [24]
2. Generate the Client Secret [25]
3. Add the Access Policy [27]

Generate the Client ID

1. Log in to the Azure portal.
2. Search for Azure Key Vault.
3. Click **+Add** to create a new key vault.





Create key vault

Instance details

Key vault name *	testkeyvaultrangerkms
Region *	East US
Pricing tier *	Standard

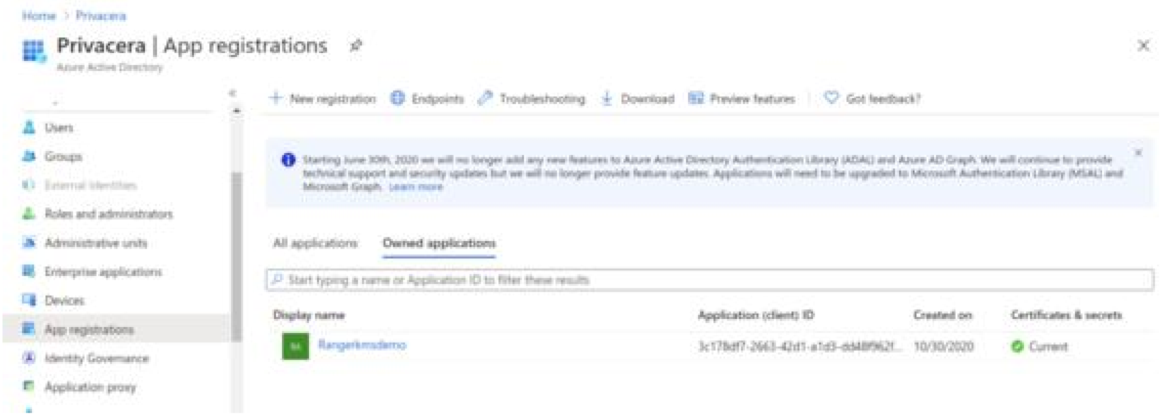
Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored within the key vault.

To enforce a mandatory retention period and prevent the permanent deletion of key vaults or secrets prior to the retention period elapsing, you can turn on purge protection. When purge protection is enabled, secrets cannot be purged by users or by Microsoft.

Soft-delete Enabled

4. After the vault is created, select the **Overview** from the navigation menu and note the Vault URL: `AZURE_KEYVAULT_URL`.
5. To connect to the vault, we need to create an application registration through the app registration.
6. Register the application. For example: rangerkmsdemo.



Privacera | App registrations

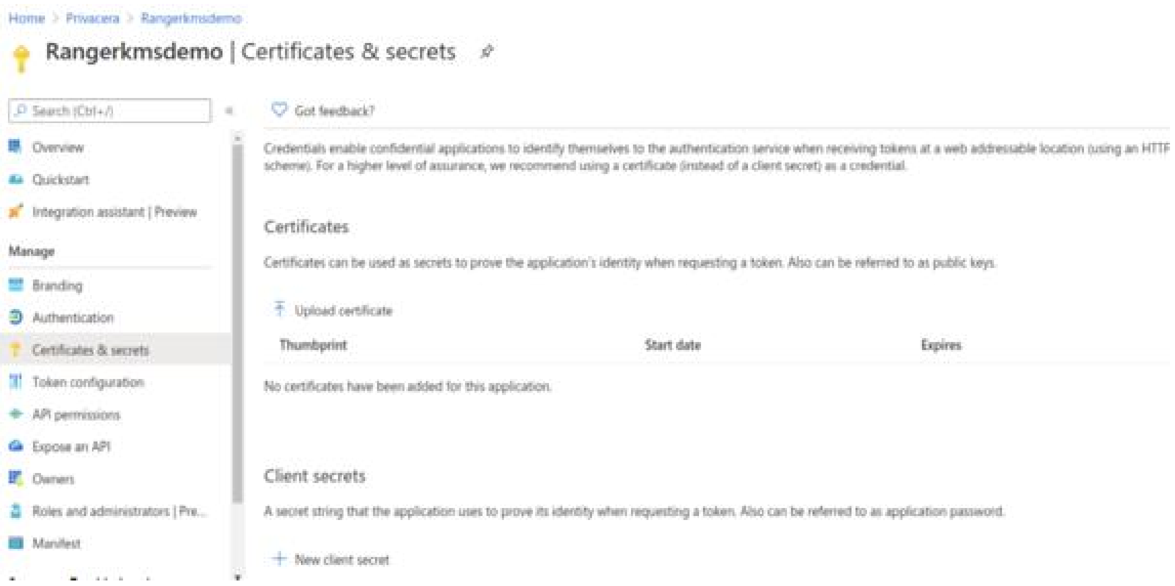
Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure AD Graph. We will continue to provide technical support and security updates but we will no longer provide feature updates. Applications will need to be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph. Learn more

Display name	Application (client) ID	Created on	Certificates & secrets
Rangerkmsdemo	3c178df7-2663-42d1-a1d3-dd48f962f...	10/30/2020	✓ Current

7. Click the registered application and in the left menu, navigate to the **Overview** section.
8. Note the **Application (client) ID** which is the `AZURE_CLIENT_ID` for connecting.

Generate the Client Secret

1. In the application screen, click on **Certificates & Secrets** in the left menu.



Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTP scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Certificates

Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

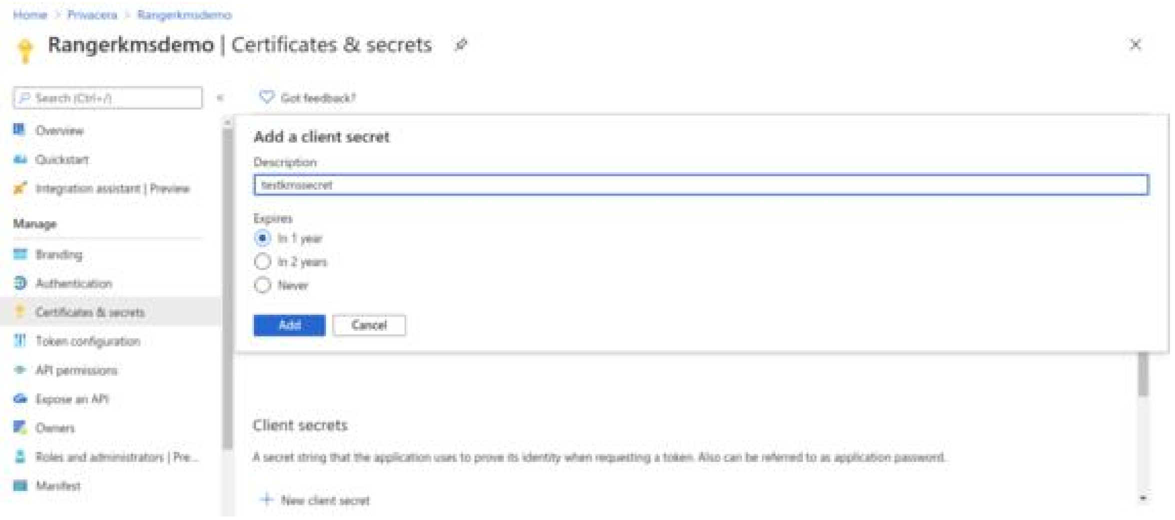
Thumbprint	Start date	Expires

No certificates have been added for this application.

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

2. Create a new client secret as shown in the example below:



Add a client secret

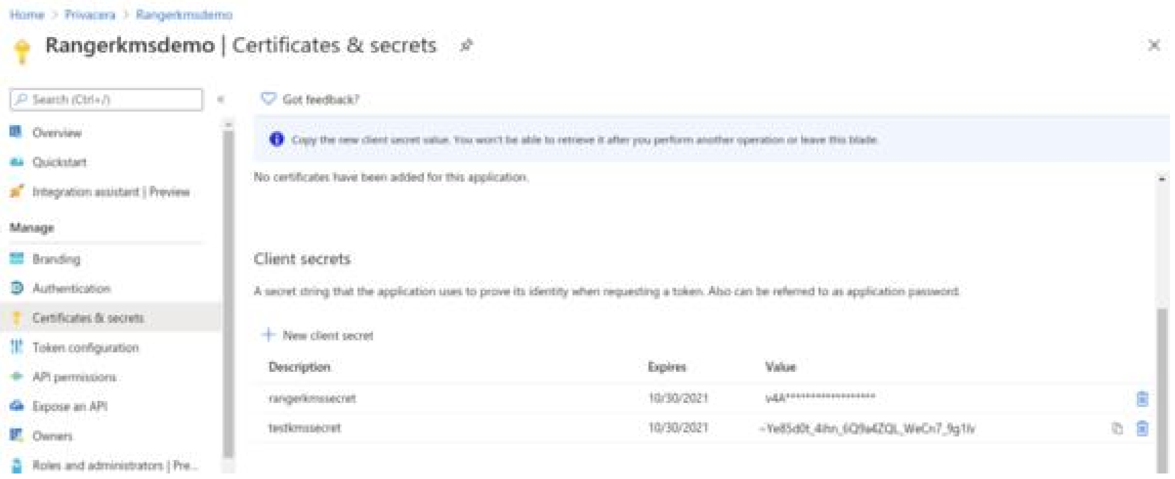
Description: testkmssecret

Expires: [✓ In 1 year] In 2 years Never

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

3. The **Client Secret** as shown - the secret value is the `AZURE_CLIENT_SECRET`.



Copy the new client secret value. You won't be able to retrieve it after you perform another operation or leave this blade.

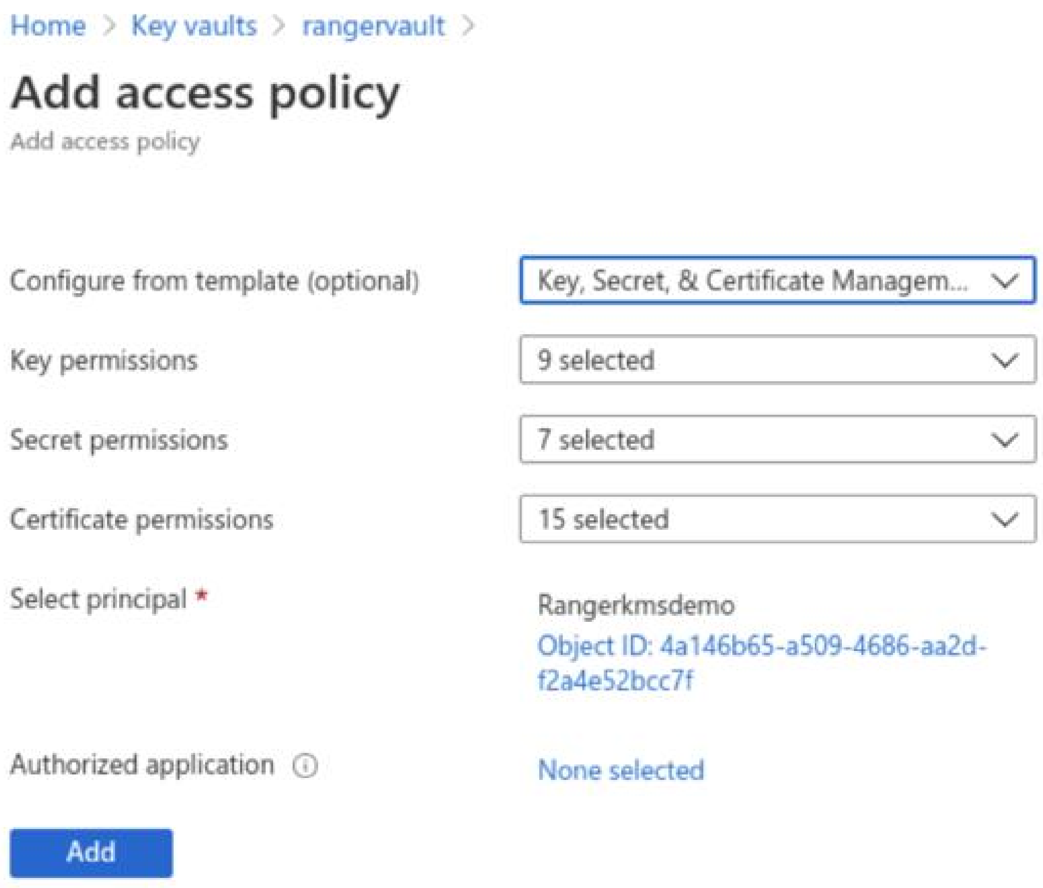
No certificates have been added for this application.

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

Description	Expires	Value
rangerkmssecret	10/30/2021	v4A********************
testkmssecret	10/30/2021	~Ye85d0t_4ihn_6Q9a4ZQL_WeCn7_9g1lv

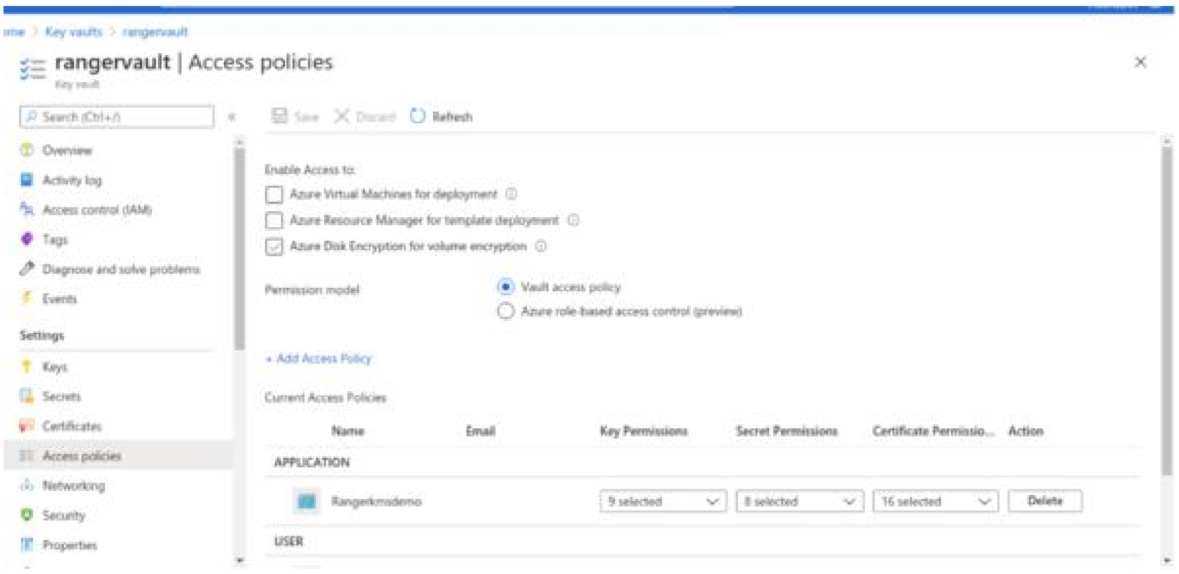
4. Next, go the key vault that was created in Step 3.
5. Select **Access Policies> +Add Access Policy**.



Configure from template (optional): Key, Secret, & Certificate Managem...
Key permissions: 9 selected
Secret permissions: 7 selected
Certificate permissions: 15 selected
Select principal *: Rangerkmsdemo
Object ID: 4a146b65-a509-4686-aa2d-f2a4e52bcc7f
Authorized application: None selected

Add the Access Policy

1. In the **Add access policy** screen, we need to set permissions to access the vault with the application that was created.
2. Select the Key permissions (mandatory), Secret permissions (optional), and Certificate permissions (optional).
3. For **Select principal** , select the application you created.



rangervault | Access policies

Enable Access to:

- [ ] Azure Virtual Machines for deployment
- [ ] Azure Resource Manager for template deployment
- [✓] Azure Disk Encryption for volume encryption

Permission model: [✓ Vault access policy] Azure role-based access control (preview)

Current Access Policies

Name	Email	Key Permissions	Secret Permissions	Certificate Permissio...	Action
APPLICATION					
Rangerkmsdemo		9 selected	8 selected	16 selected	Delete
USER					

4. Go to `Privacera/docker/ranger/kms/install.properties` and change the following values:

```
AZURE_KEYVAULT_ENABLED=true
AZURE_KEYVAULT_SSL_ENABLED=false
AZURE_CLIENT_ID=(from step 3.3)
AZURE_CLIENT_SECRET=(from step 3.6)
#AZURE_AUTH_KEYVAULT_CERTIFICATE_PATH (mandatory field. Value can be None/dummy)
AZURE_AUTH_KEYVAULT_CERTIFICATE_PATH=/home/machine/Desktop/azureAuthCertificate/keyva
# AZURE_AUTH_KEYVAULT_CERTIFICATE_PASSWORD (mandatory field. Value can be None/dummy)
AZURE_AUTH_KEYVAULT_CERTIFICATE_PASSWORD=certPass AZURE_MASTERKEY_NAME=RangerMasterKe
# E.G. RSA, RSA_HSM, EC, EC_HSM, OCT
AZURE_MASTER_KEY_TYPE=RSA
# E.G. RSA_OAEP, RSA_OAEP_256, RSA1_5, RSA_OAEP
ZONE_KEY_ENCRYPTION_ALGO=RSA_OAEP
AZURE_KEYVAULT_URL=(from step 4 )
```

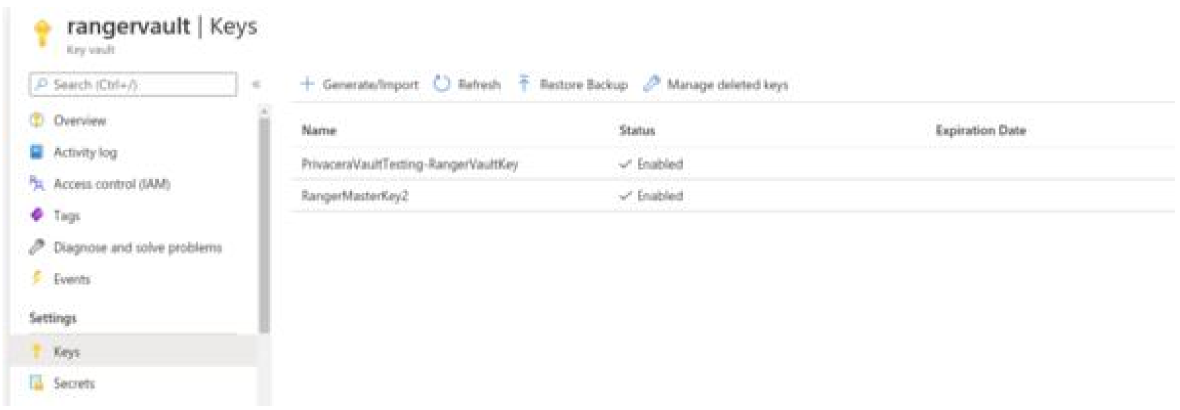
NOTE

The fields that say `Value can be null/dummy` must have some value - cannot be blank.

5. Restart Ranger KMS.

```
cd ~/privacera/docker
./privacera_services restart ranger-kms
```

6. The master key is created when Ranger KMS is restarted. Verify that the master key (name that is set in the properties) is created in the vault under Keys:



Name	Status	Expiration Date
PrivaceraVaultTesting-RangerVaultKey	✓ Enabled	
RangerMasterKey2	✓ Enabled	

When the Client ID and Client certificate are added and the Ranger KMS is restarted, an error occurs in the KMS logs: `~/privacera/docker/logs/ranger/kms/`.

7. Exit the container and restart Ranger KMS.

Migrate Ranger KMS master key on Privacera Platform

The following steps will migrate the master key of Ranger KMS from its database to the Azure Key Vault.

1. Run the following commands to enter the Ranger KMS shell:
   - **Docker shell**

     ```
     cd /home/ec2-user/privacera/docker
     ./privacera_services shell ranger-kms
     ```
   - **Kubernetes shell**

     In the variable, <NAMESPACE>, provide your namespace.

     ```
     kubectl get pods -n <NAMESPACE>
     kubectl exec -it <ranger_kms_pod_name> -n <NAMESPACE> -- bash
     ```
2. Run the following commands to run the migration script:

   ```
   bash DBMKTOAZUREKEYVAULT.sh <azureMasterKeyName> <azureMasterKeyType>
   <zoneKeyEncryptionAlgo> <azureKeyVaultUrl> <azureClientId> <isSSLEnabled:true/false>
   <clientSecret / Certificate Path>
   ```

Parameter	Description
`<azureMasterKeyName>`	Name of the Master Key you want to migrate.
`<azureMasterKeyType>`	Type of the Master Key. For example, RSA
`<zoneKeyEncryptionAlgo>`	Encryption algorithm used in the Master Key. For example: RSA_OAEP
`<azureKeyVaultUrl>`	Azure Key Vault URL.
`<azureClientId>`	Azure Client ID.
`<isSSLEnabled:true/false>`	Enable SSL. For example: `true`
`<clientSecret / Certificate Path>`	If the authentication is done without SSL enabled, get the client secret. For more information, click here [24]. If the authentication is done with SSL enabled, get the certificate secret. For more information, click here [19].

Ranger KMS with Azure Key Vault on Privacera Platform

For information about working with the Ranger Key Management System (KMS) and the Azure Key Vault (AKV) on Privacera Platform, see:

- Connect to Azure Key Vault with Client ID and Client Secret on Privacera Platform [24]
- Connect to Azure Key Vault with a client ID and certificate on Privacera Platform [19]

- Migrate Ranger KMS master key on Privacera Platform [29] to Azure Key Vault.

Schemes

Privacera Encryption relies on *schemes*. A *scheme* is a combination of *formats*, *algorithms*, and *scopes*. There are three types of schemes:

- **Encryption schemes [31]**: schemes that encrypt or decrypt the data.
- **Presentation schemes [43]**: optional schemes that obfuscate decrypted data to a form suitable to display to authorized users.
- Masking schemes [44]: schemes that permanently transform the data one-way.

All schemes rely on the same set of encryption formats, algorithms, and scopes:

- **Format**: defines the data type and structure to be encrypted, such as alphanumeric, credit card, email address, or social security number.
- **Algorithm**: specifies the mathematics used to encrypt, such as AES, FPE, or SHA.
- **Scope**: defines the extent of the data encryption, such as the first four digits, an IP domain, or all data. Scoping ALL is recommended.

A *scheme policy* defines access control: users who have permission to access a scheme.

For example, you might rely on a Privacera-supplied encryption scheme to protect a PII field called "EMAIL" with the following properties:

- Uses `EMAIL` format
- Applies the SHA-256 algorithm for a one-way hash
- Is scoped with "masked domain" to hide the portion of the email to the right of the @ sign

You can also define your own custom encryption, presentation, and masking schemes.

Encryption schemes

Encryption schemes are schemes that encrypt and decrypt your data.

IMPORTANT

When using encryption schemes, make sure to:

- Keep a record of which schemes you use to encrypt or transform which data. You need to use the same scheme to decrypt that data.
- Protect your active schemes. Consider exporting them to a secure location.
- Never delete your active schemes.

Otherwise, you will be unable to decrypt your data.

View encryption schemes

To view your encryption schemes, select **Encryption & Masking** > **Schemes** from the Privacera Portal navigation menu.

You can import, export, define new encryption schemes, or modify existing ones.

Default encryption schemes

The following is a list of the Privacera-supplied system encryption schemes, which are enabled by default. The name of a scheme in general describes the type of data the scheme is designed to encrypt.

- SYSTEM_US_PHONE_FORMATTED
- SYSTEM_ACCOUNT
- SYSTEM_PERSON_NAME
- SYSTEM_SSN
- SYSTEM_EMAIL
- SYSTEM_ADDRESS
- SYSTEM_CREDITCARD

Default encryption schemes for APIs

Depending on the API you're using, you can use some default schemes [31] but not others. For more information, see the following topics.

- Default encryption schemes for the Privacera API [32]
- Default encryption schemes for the Bouncy Castle API [42]

NOTE

Schemes not listed in the Privacera docs are deprecated. See Deprecated encryption schemes [49] for more information.

NOTE

For a scheme with a numeric format type and FPE algorithm, the numeric data on the calls to `/protect` and `/unprotect` must be a string.

To preserve the format and length, the encrypted output must also be stored as a string.

Example: Encrypting a number via FPE like 123456 might result in output like 027931.

Default encryption schemes for the Privacera API

The following encryption schemes can be used with the Privacera API. The request and response examples show the `datalist` JSON array.

- Format: Alphanumeric [33]
- Format: ASCII [33]
- Format: CC [34]
- Format: Driver License [40]
- Format: Email [38]
- Format: FPE_ALPHA_NUMERIC [41]
- Format: HASHING [38]
- Format: Host/Domain [39]
- Format: IP [41]
- Format: LITERAL [34]

- Format: Numeric [35]
- Format: SSN [37]
- Format: Text [37]
- Formats: DATE and Date_DD_MM [36]

Format: Alphanumeric

- Algorithm: Alphanumeric, Scope: All

Example

- Request:

```
"datalist": [
    [
        "TUCSON AZ 85705,USA",
        "testdata115",
        "105 Sikes Hall, Clemson, SC 29634, USA",
        "177A Bleecker Street",
        null
    ]
]
```

- Response:

```
"datalist": [
    [
        "hGL8f5ycfDDrxguRRZhDTPINOfHZmlxik5bW2xz9Mbg=",
        "7GEWk9XuIigzkTczc9Ntzg==",
        "9e6obWu6mh9vK2xkEcFvOeXSYwve2Ws9jQ1AEBVxc3zj5lFGNcBPxxLpgyyZin0u",
        "MDtays2tyyOv5egH+OXbk9UqL2RDTZRhqNYxaEULsjs=",
        null
    ]
]
```

Format: ASCII

7-bit ASCII character set, excluding control characters.

- Algorithm: FPE, Scope: All

Example

- Request:

```
"datalist": [
    [
        "testvalue123",
        "This is a sample text",
        "This is sample 123 alphanumeric text 123.",
        "123456789098765",
        "123!@#R)(*&^4567JHG",
        null
    ]
]
```

- Response:

```
"datalist": [
    [
       "[pa&xA_)1qs=",
        "?xUs.H';NIy>BJ0@y9{qq",
        "o/|R7&k)d>dmp^Am}.%-F]_Ym7c]@B~Xm)eOB+=w*",
        "/apLBEweK)?| *t",
        "{lD+U%cMLKM]k+`lt}.",
        null
      ]
  ],
```

Format: LITERAL

Free-form: no specific format required.

- Algorithm: FPE, Scope: All
- Algorithm: Standard 256-bit, Scope: All

Example of FPE, All

- Request:

```
"datalist": [
    [
        "TRUE",
        "FALSE",
        "123876.0988",
        "123876",
        "Literal",
        "Test123Text",
        null
    ]
]
```

- Response:

```
"datalist": [
    [
        "",
        "",
        "",
        "",
        "",
        "",
        null
    ]
]
```

Format: CC

Credit card. Numeric from 14 to 19 digits. Hyphens and spaces allowed.

- Algorithm: FPE, Scope: All
- Algorithm: FPE, Scope: First 4 digits

- Algorithm: FPE, Scope: Last 4 digits

Example of FPE, All

- Request:

```
"datalist": [
    [
        "236864479139819",
        "160201209940524",
        "41228020889831",
        "7529274609013685",
        null,
        "6536921047107462",
        "4766530513049409"
      ]
  ]
```

- Response:

```
"datalist": [
    [
        "524312768689370",
        "535332579591178",
        "79759512315352",
        "1072002057261056",
        null,
        "3907516129227718",
        "6712017221140690"
    ]
]
```

Format: Numeric

Digits from 0 through 9.

- Algorithm: FPE, Scope: All

Example

- Request:

```
"datalist": [
    [
        null,
        "a9876543211098",
        "9876543211098",
        "a9876543211098",
        "acn9876543211098",
        null,
        "1234567890897654321",
        "1ab4 df56 7qwer2343",
        "1234543 5434 23454",
        "priv9876543211098",
        "acn9876543211098",
        null
        ]
    ]
```

- Response:

```
    "datalist": [
        [
            null,
            "a8440422448831",
            "5980689261168",
            "a8440422448831",
            "acn1390446821808",
            null,
            "39634136093054120 90",
            "4ab6 df25 4qwer6711",
            "3073815 5226 34978",
            "priv1617217642784",
            "acn1390446821808",
            null
        ]
    ]
```

- Algorithm: FPE, Scope: All

Formats: DATE and Date_DD_MM

- Algorithm: FPE, Scope: All

For details on allowable formats, see Date input formats and ranges.

Example

- Request:

```
    "datalist": [
        [
            "16/12/3352",
            "09/02/3508",
            "16-12-3352",
            "21-03-3421",
            "19/12/3224 21:01:24:202",
            null
        ]
    ]
```

- Response:

```
       "datalist": [
        [
            "16/12/3352",
            "09/02/3508",
            "16-12-3352",
            "21-03-3421",
            "12/07/3871 20:44:36:480",
            null
        ]
    ]
```

Format: SSN

US Social Security Number. Nine digits. Hyphens and spaces allowed.

- Algorithm: FPE, Scope: Last 4 digits
- Algorithm: FPE, Scope: All

Example of FPE, All

- Request:

```
"datalist": [
    [
        "778-62-8144",
        "030 72 7381",
        "709066491",
        "163254042",
        null,
        "805 14 1893",
        "401318448"
    ]
]
```

- Response:

```
"datalist": [
    [
        "932-88-1456",
        "828 92 5898",
        "954061516",
        "998726200",
        null,
        "980 21 5905",
        "191897078"
    ]
]
```

Format: Text

- Algorithm: FPE, Scope: All

Example of FPE, All

- Request:

```
"datalist": [
    [
        "778-62-8144",
        "030 72 7381",
        "709066491",
        "163254042",
        null,
        "805 14 1893",
        "401318448"
    ]
]
```

- Response:

```
"datalist": [
    [
        "932-88-1456",
        "828 92 5898",
        "954061516",
        "998726200",
        null,
        "980 21 5905",
        "191897078"
    ]
]
```

Format: HASHING

Same as Format: ASCII [33]. These are one-way hashes.

- Algorithm: SHA-256, Scope: All
- Algorithm: SHA-512, Scope: All

Example of SHA-256, All

- Request:

```
"datalist": [
    [
        "8743b52063cd84097a65d1633f5c74f5",
        "hashvalue115",
        "Test123Text",
        null
    ]
]
```

- Response:

```
"datalist": [
    [
        "74ee1fae245edd6f27bf36efc3604942479fceefbadab5dc5c0b538c196eb0f1",
        "492c94273948d5140dcfef60b15a99b9c2cd5e730a5d40d2991548255825d473",
        "c9ecc7cecff05b064da8a89befa266e84da87409a7d8624ec15252affb70d732",
        null
    ]
]
```

Format: Email

Must include @ sign.

- Algorithm: FPE, Scope: All
- Algorithm: FPE, Scope: masked username
- Algorithm: FPE, Scope: masked domain

Example of FPE, All

- Request:

```
"datalist": [
    [
        "test@domain.com",
        "lastname@domain.com",
        "test.email.with+symbol@domain.com",
        "id-with-dash@domain.com",
        "example-abc@abc-domain.com",
        "admin@mailserver1",
        "#!$%&'*+-/=?^_{}|~@domain.org",
        "example@localhost",
        "example@s.solutions",
        "test@com",
        "test@localserver",
        null
    ]
]
```

- Response:

```
"datalist": [
    [
        "T~oi@GaRxEU.ZFq",
        "R82`Rs7E@GaRxEU.ZFq",
        "s%x{.&FEi!.qPEjpST2gK#@GaRxEU.ZFq",
        "t+g_4s+Vn_?7@GaRxEU.ZFq",
        "bPVRw9_x_J`@DmF-AyWNGj.gxA",
        "BnAIk@lhGbMXvogj1",
        "GZhp3&iMy^X|0Jij%s@WCXdsf.BYi",
        "vUnO=Fb@IWyJfKkFW",
        "vUnO=Fb@X.wpkHRwTbu",
        "T~oi@nZF",
        "T~oi@EfwTCYFFfgu",
        null
    ]
]
```

Format: Host/Domain

Internet standard domain name, or portion thereof, with periods.

- Algorithm: FPE, Scope: All

Example

- Request:

```
"datalist": [
    [
        "cornell.edu",
        "www.google.com",
        "en.wikipedia.org",
        ".com",
        "www.privacera.com",
        "www.privacera.com",
        ".edu",
        "10.211.95.191",
        null
    ]
]
```

- Response:

```
"datalist": [
    [
        "uf8T8tY.u54",
        "1Wr.f6NCmk.M9m",
        "fj.dbwLIn9DR.BfV",
        ".qCB",
        "XGY.GPRNgo1Wo.x7t",
        "XGY.GPRNgo1Wo.x7t",
        ".B56",
        "y4.VTB.Uh.V2H",
        null
    ]
]
```

Format: Driver License

- Algorithm: FPE, Scope: All

Example

- Request:

```
"datalist": [
    [
        "A123456789012",
        "12345678X",
        null,
        "123456789",
        "m1234567",
        "12345678123456789",
        "123456789",
        null,
        "113654424",
        "999000680",
        "B13654424",
        "G544-061-73-925-0",
        "AA123456Z",
        null
    ]
]
```

- Response:

```
"datalist": [
    [
        "09HnovI2QR9jw",
        "pIJijAhlj",
        null,
        "pnZaDghd0",
        "ICSdAHiD",
        "92SRB3QE5S6TunSRA",
        "pnZaDghd0",
        null,
        "J1XT5UuBq",
        "SHdt78Two",
        "PMGoghnkh",
        "rusP-R4U-EG-nVV-r",
        "YslNiR2As",
        null
    ]
]
```

Format: FPE_ALPHA_NUMERIC

- Algorithm: Alphanumeric, Scope: All
- Algorithm: FPE, Scope: All
- Algorithm: Standard, Scope: All
- Algorithm: Standard 256-bit, Scope: All

Example of FPE, All

- Request:

```
    "datalist": [
        [
            "Acc965121354",
            "testdata123samplevalue",
            "sample value 2nd instance",
            "221, baker street",
            null
        ]
    ]
```

- Response:

```
    "datalist": [
        [
            "4eOPie2yXN1f",
            "SsGfMkh12uH1ndQnsDaa1V",
            "j8pHr5 CdFLR LUc 0zw1ZuhK",
            "om5, KUR9R bBjjd2",
            null
        ]
    ]
```

Format: IP

Internet Protocol v4 or v6 standard address

- Algorithm: FPE, Scope: All

Example

- Request:

```
    "datalist": [
    [
        null,
        "10.211.95.191",
        "ABCD:EF01:2345:6789:ABCD:EF01:2345:6789",
        "2001:DB8:0:0:8:800:200C:417A",
        "123.123.12.1",
        null,
        "0.0.0.0",
        "10.31.31.54",
        null
    ]
]
```

- Response:

```
"datalist": [
    [
        null,
        "184.54.42.61",
        "ABCD:EF6a:e277:216a:ABCD:EFf9:5b8c:3a24",
        "9623:DB5:5:6:4:b3a:34cC:9ecA",
        "33.71.6.126",
        null,
        "223.195.44.37",
        "138.217.142.157",
        null
    ]
]
```

Default encryption schemes for the Bouncy Castle API

The following table displays the default encryption schemes that can be used with the Bouncy Castle API. To learn more about the Bouncy Castle API, see the Bouncy Castle documentation.

The only allowable scope for schemes that use the Bouncy Castle API is All.

Table 1. Privacera-supplied encryption schemes for the Bouncy Castle API

Format	Algorithm	Scope
Alphanumeric	• AES 128 • AES 256	All
ASCII	• AES 128 • AES 256	All
CC	• AES 128 • AES 256	All
Date	• AES 128 • AES 256	All
DateTime	• AES 128 • AES 256	All
Email	• AES 128 • AES 256	All
Host/Domain	• AES 128 • AES 256	All
IP	• AES 128 • AES 256	All
Numeric	• AES 128 • AES 256	All

Format	Algorithm	Scope
SSN	• AES 128 • AES 256	All
Text	• AES 128 • AES 256	All

Create custom encryption schemes

In addition to Privacera's default encryption schemes, you can also create your own custom encryption schemes.

Prerequisites:

Before you create a custom encryption scheme, prepare the following details:

- A useful name for the encryption scheme
- A description of the encryption scheme
- The names of the tags that you want to encrypt
- The data format, datatype, algorithm, and scope that you want to apply. See Schemes [31] for more information.

To create custom encryption schemes in the Privacera Portal, follow these steps:

1. From the navigation menu, select **Encryption & Masking** > **Schemes**.
2. Click **Add** to add a new scheme.
   The Add Encryption Scheme dialog displays.
3. Enter the following details into the respective fields:
   - **Name**: the name of the scheme, such as US_PHONE_3rdParty
   - **Description**: a description of the scheme
   - **Encryption API**: PRIVACERA (default) or BOUNCY_CASTLE
   - **Format type**: the encryption format type, such as FPE_ALPHA_NUMERIC, alphanumeric
   - **Scope**: all
   - **Value**: the value of the scheme
   - **Algorithm**: the encryption algorithm: FPE, Hash, Token, Mask, Standard 256, SHA_256, SHA_512
4. Click **Save**.
   The encryption scheme is created.

Numeric formats with FPE algorithm

For a scheme with Numeric format type and FPE algorithm, the numeric data on the calls to `/protect` and `/unprotect` must be a string.

To preserve the format and length, the encrypted output must also be stored as a string.

Example: Encrypting a number via FPE like `123456` might result in output like `027931`.

Presentation schemes

A presentation scheme controls how decrypted data should be presented to authorized users. After data is decrypted, a presentation scheme can obfuscate it *again* in a form suitable for displaying to those authorized users.

- If no presentation scheme is specified, the decrypted data is viewable in its original form.
- If the user does not have access rights to view the data, an error message is displayed. For information about granting user access to schemes, see Scheme policies [45].

Default presentation schemes

The following is a list of the Privacera-supplied system presentation schemes. The name of a scheme in general describes the type of data the scheme is designed to encrypt.

The formats, algorithms, and scopes associated with each scheme are described in Default encryption schemes [32].

- SYSTEM_PRESENTATION_PERSON_NAME
- SYSTEM_PRESENTATION_SSN
- SYSTEM_PRESENTATION_EMAIL
- SYSTEM_PRESENTATION_ADDRESS
- SYSTEM_PRESENTATION_CREDITCARD
- SYSTEM_PRESENTATION_US_PHONE_FORMATTED
- SYSTEM_PRESENTATION_ACCOUNT

View default presentation schemes

To see the system presentation schemes:

1. Expand **Encryption & Masking**.
2. Click **Schemes**.
3. Click the **Presentation Scheme** tab.

You can import, export, define new presentation schemes or modify existing ones.

Create custom presentation schemes

Before defining a scheme, plan what you want it to do. See Default encryption schemes [32].

1. On the left, expand **Encryption & Masking**.
2. On the left, click **Schemes**.
3. Click the **Presentation Scheme** tab.
4. To add a new presentation scheme, click **+ Add**.
5. If you are using Privacera Platform, click the **Presentation Scheme** tab.
6. Fill in the following fields:
   - **Name**: a useful name for this scheme.
   - **Description**: a helpful description of what the scheme does.
   - **Format Type**: From the pulldown, select the format you want to use.
   - **Scope**: From the pulldown, specify the extent of the data transform. The available scopes depend on which format you have chosen.
   - **Value**: This optional field is for certain types of scopes.
   - **Algorithm**: Select the required algorithm, which depends on the format you have chosen.
7. Click **Save** to retain your changes or **Close** to discard them.

Masking schemes

Masking schemes are one-way transformations of data that do not allow for decryption. Once a masking scheme is applied, the original data is completely replaced and cannot be unmasked.

Masking techniques

There are two different techniques that a masking scheme can use to mask your data:

- **Nullify**: the original string is completely removed
- **Redaction**: the original string is overwritten with a masking character. You can specify a masking character, or use the default **x**.
  You can redact a string with a masking character that is repeated five times, or you can retain the format and length of the original string. This preserves all of the special characters in the original string and replaces the alphanumeric characters with the masking character.

For example:

- **Original string:**`somebody@BigCo.com`
- **Result without maintaining format and length:**`xxxxx`
- **Result with maintaining format and length:**`xxxxxxxx@xxxxx.xxx`

Masking with the Encryption REST API

Masking schemes use the `/protect` REST API endpoint. Input to `/protect` must be in JSON format.

Because masking is one-way, do not use masking schemes with the `/unprotect` endpoint. Using a masking scheme with `/unprotect` returns an error.

You can combine masking and encryption in a single API request, so that you encrypt some fields and mask other fields at the same time.

Create custom masking schemes

You can create custom masking schemes to use with the encryption REST API.

Prerequisites:

- Choose a name for the masking scheme that is easy to remember.
- Think of a helpful description for the masking scheme.
- Decide which format [48] you want to use for the masking scheme.
- Decide if the masking scheme should use the nullify or redaction masking technique.
- If the scheme is to redact:
  - Decide on a suitable masking character to replace the original characters.
  - Decide if you want to retain the original string's format and length.

To create a custom masking scheme, follow these steps:

1. From the navigation menu, select **Encryption & Masking** > **Schemes**.
2. Click the **Masking Scheme** tab.
3. Click **Add**.
   The Add Masking Scheme dialog displays.
4. In the **Name** field, enter a name for the masking scheme.
5. In the **Description** field, enter a description of the masking scheme (optional).
6. From the **Format Type** dropdown, select a format.
7. In the **Choose Masking Technique** section, select either **Nullify** or **Redaction**.
8. If you chose **Redaction**, fill out the following details in the **Redaction Settings** section:
   a. In the **Masking Character** field, enter a masking character or use the default **x**.
   b. If you want to maintain the original formatting and length of the masked data, select **Maintain original formatting and length**.
9. Click **Save**.
   The masking scheme is created.

Scheme policies

Through the use of scheme policies, data access administrators can restrict data users or groups in their use of specific encryption or presentation schemes.

Access Management > Scheme Policies is part of the Privacera Encryption service and is enabled when the **Privacera Encryption Gateway (PEG)** service is added and configured. Its layout, organization, and functions are analogous to Resource policies.

A scheme service is a set of scheme oriented access and usage policies. The **privacera_peg** scheme service contained in the PEG **service group** is automatically created when the PEG service is enabled.

The *privacera_peg* service contains a set of ***scheme policies***, which are the means to scope use of encryption and presentation schemes to individual or groups of data users.

Click **privacera_peg** to access and manage a list of existing scheme policies and to add and define new policies.

As with Resource Policies, each scheme policy has a Name, Description, associated Labels and access (usage) control rules grouped by setting Allow and Deny conditions and exceptions to Allow and Deny. The difference is that the target of control for Scheme policies are Encryption Schemes and Presentation Schemes rather than data resources.

Plan for scheme policies

Before creating scheme policies, consider the following:

- Ensure that you have created the users, groups, or roles whose access to the PEG API endpoints you want to control.
- Decide on a useful name for the scheme policy and a useful description of it.
- Decide if you want the scheme policy to be in effect for only a certain validity time period.
- Decide how you want to provide access:
  - Give access to all roles or groups but deny access to specific other roles, groups, or users.
  - Deny access to all roles or groups but give access to specific other roles, groups, or users.
- Decide if you want to use admin delegation for the specific users so that a service user can make PEG REST API endpoints on their behalf.

Create scheme policies

You can create scheme policies to restrict data users or groups in their use of specific encryption or presentation schemes.

Create scheme policies on Privacera Platform

On Privacera Platform, you can create scheme policies that provide certain users access to the encryption and masking schemes described in Encryption schemes [31] and Masking schemes [44].

Add user in default policy

To create a scheme policy, you need to add a `privacera_service_discovery` user in the default **privacera_peg** policy using the Privacera Portal.

To add the user, follow these steps:

1. From the navigation menu, select **Access Management** > **Scheme Policies**.
2. In the **PEG** section, click **privacera_peg**.
3. In the **peg** policy row, click the **Edit** icon.
4. In the **Allow Conditions** section, search for and select the `privacera_service_discovery` user from the **Select User** dropdown menu.

Create scheme policies on Privacera Platform

You can create scheme policies using the Privacera Portal.

To create scheme policies, follow these steps:

1. From the navigation menu, select **Access Management** > **Scheme Policies**.
2. In the **PEG** section, click **privacera_peg**.
3. Click **Add New Policy**.
4. If you want this scheme policy to be in effect for a specific time period, click **Add Validity Period** and enter the details into the **Policy Validity Period** dialog.
   The **Enabled** toggle allows you to create a policy.

5. In the **Policy Name** text box, enter a name for the scheme policy.
6. From the **Policy Labels** dropdown menu, select the labels that you want to to the scheme policy.
7. In the **Encryption scheme** dropdown, you can select either the **Encryption scheme** or the **Masking scheme**.
   - If you select Encryption Scheme, enter the name of the encryption scheme to which you want to apply the policy. Repeat this for all required encryption schemes.
     When you select the Encryption scheme, the Presentation Scheme appears automatically.
   - If you select Masking Scheme, enter the name of the masking scheme to which you want to apply the policy. Repeat this for all required masking schemes.
     When you select Masking Scheme, the `mask` permission appears in the Permissions section.
8. From the **Presentation Schemes** dropdown menu, select the name of the presentation scheme that you want to apply the policy to. Repeat this for all required presentation schemes.
9. In the **Description** field, enter a description of the scheme policy.
10. If you do not want audit logs, disable the **Audit Logging** toggle.

Define allow conditions

In the **Allow Conditions** section, you can select the names of roles, groups, or users that you want to give access to the schemes.

To define the allow conditions for a scheme policy, follow these steps:

1. In the **Allow Conditions** section, search and select the `privacera_service_discovery` user from the **Select User** dropdown menu.
2. Select the names of roles, groups, or users you want to give access to the schemes.
3. In the **Permissions** column, click **Add Permission** and specify the permission you want to give.
4. In the **Delegate Admin** column, select the checkbox if you want the service user to be able to make API endpoints on behalf of the application user.
5. In the **Exclude from Allow Conditions** section, select the roles, groups, or users that you want to exclude from the allow conditions.

Define deny conditions

In the **Deny Conditions** section, you can select the roles, groups, or users that you want to deny access to the schemes.

To define the deny conditions for a scheme policy, follow these steps:

1. In the **Deny Condition** section, click **Add Permission** and specify the permission you want to deny.
2. Click **Delegate Admin** if you want the service user to be able to make API endpoints on behalf of the application user.
3. If you want to deny access to specific roles, groups, or users, under **Exclude from Deny Conditions**, select those roles, groups, or users you want to exclude.
4. In the **Exclude from Deny Conditions** section, select the roles, groups, or users that you want to exclude from the deny conditions.
5. Click **Save** to save the scheme policy.

Create scheme policies on PrivaceraCloud

Prerequisites:

- Decide which users, groups, or roles you want to apply the scheme policy to.
- Decide how you want to give permissions to those users, groups, or role. You can either allow them or deny them the use of encryption or presentation schemes.

To create scheme policies on PrivaceraCloud, follow these steps:

1. Go **Access Management > Scheme Policies**.
2. Find and click the PEG resource name.
3. If you want this scheme policy to be in effect for only a specific time period, in the upper right, click **Add Validity Period** and enter the details of that period.
4. Enter a name for the scheme policy.
5. If desired, select any policy labels you want for this scheme policy.
6. If you do not want audit logging, use the **Audit Logging** toggle to turn it off.
7. With the **Enabled** toggle, you can create the policy but not yet enable it.
8. In the **Encryption scheme** dropdown, you can select either the **Encryption scheme** or the **Masking scheme**.
   - If you select Encryption Scheme, enter the name of the encryption scheme to which you want to apply the policy. Repeat this for all required encryption schemes.
     When you select the Encryption scheme, the Presentation Scheme appears automatically.
   - If you select Masking Scheme, enter the name of the masking scheme to which you want to apply the policy. Repeat this for all required masking schemes.
     When you select Masking Scheme, the `mask` permission appears in the Permissions section.
9. For **Presentation Schemes** field, select the name of the presentation scheme to which you want to apply the policy. Repeat this for all required presentation schemes.
10. Enter a description of this scheme policy.
11. Depending on how you have decided to allow or deny access, use the fields in **Allow conditions** or **Deny Conditions** and follow the corresponding steps below:
    - Allow Access
      a. Under **Allow Conditions**, in the role, group, and user fields, select the names of roles, groups, or users you want to give access to the schemes.
      b. On the right, click **Add Permission** and specify the permission you want to give.
         - **getSchemes** permission is required for Databricks UDFs.
         - **protect/unprotect** permissions are required for REST API calls.
      c. On the far right, click **Delegate Admin** if you want the service user to be able to make API requests on behalf of the application user.
      d. If you want to deny access to specific roles, groups, or users, under **Exclude from Allow Conditions**, select those roles, groups, or users you want to exclude.
    - Deny Access
      a. Under **Deny Conditions**, in the role, group, and user fields, select the names of roles, groups, or users you want to deny access to the schemes.
      b. On the right, click **Add Permission** and specify the permission you want to deny.
      c. On the far right, click **Delegate Admin** if you want the service user to be able to make API requests on behalf of the application user.
      d. If you want to give access to specific roles, groups, or users, under **Exclude from Deny Conditions**, select those roles, groups, or users you want to allow.
12. Save your scheme policy.

Formats

A *format* is the data type and structure of the input data to be encrypted.

For example, the format of your input data could be:

- Numeric
- Date
- Credit card
- Social security number

Algorithms

Algorithms are the mathematics used to encrypt your data.

There are two types of algorithms:

- Two-way encryption and decryption
- One-way hashes

About LITERAL

One type of one-way transformation is the `LITERAL` replacement of data. This option replaces the specified data with the name of the tag associated with the data.

For example, if a database field is tagged as `PERSON_NAME`, when an encryption transform is applied as `LITERAL`, the field's value is replaced with `PERSON_NAME`.

CAUTION
If you use `LITERAL`, the original data cannot be recovered.

Scopes

Scopes define the extent of your data encryption, such as the first four digits, an IP domain, or all data.

The **ALL** scope is recommended as the most comprehensive treatment of the extent of the data. However, you can choose from other available scopes.

Deprecated encryption schemes

The following schemes are deprecated as of Privacera Platform release 6.3 and will not be supported in future releases.

Deprecated: IP

- Format Type: IP, Scope: ALL, Algorithm: AlphaNumeric

Deprecated: Host/Domain

- Format Type: Host/Domain, Scope: ALL, Algorithm: AlphaNumeric
- Format Type: Host/Domain, Scope: ALL, Algorithm: Standard
- Format Type: Host/Domain, Scope: ALL, Algorithm: Standard 256 bit

Deprecated: Text

- Format Type: Text, Scope: ALL, Algorithm: AlphaNumeric
- Format Type: Text, Scope: ALL, Algorithm: Standard
- Format Type: Text, Scope: ALL, Algorithm: Standard 256 bit

Deprecated: Driver License

- Format Type: Driver License, Scope: ALL, Algorithm: AlphaNumeric
- Format Type: Driver License, Scope: ALL, Algorithm: Hash
- Format Type: Driver License, Scope: ALL, Algorithm: Mask
- Format Type: Driver License, Scope: ALL, Algorithm: Standard
- Format Type: Driver License, Scope: ALL, Algorithm: Standard 256 bit
- Format Type: Driver License, Scope: ALL, Algorithm: SHA_256 Hash

Deprecated: LITERAL

- Format Type: LITERAL, Scope: ALL, Algorithm: AlphaNumeric
- Format Type: LITERAL, Scope: ALL, Algorithm: Standard

Deprecated: Alphanumeric

- Format Type: Alphanumeric, Scope: ALL, Algorithm: FPE
- Format Type: Alphanumeric, Scope: ALL, Algorithm: Standard
- Format Type: Alphanumeric, Scope: ALL, Algorithm: Standard 256 bit

About LITERAL

One type of one-way transformation is the `LITERAL` replacement of data. This option replaces the specified data with the name of the tag associated with the data. For example, if a database field is tagged as `PERSON_NAME`, when an encryption transform is applied as `LITERAL`, the field's value is replaced with `PERSON_NAME`.

Using `LITERAL` means that the original data cannot be recovered.

User-defined functions (UDFs)

With UDFs, you can encrypt and decrypt your data directly in your database, relying on Privacera Encryption libraries you embed in your application.

Encryption UDFs for Apache Spark on PrivaceraCloud

This section describes how to install and configure the Privacera Crypto jar in Apache Spark to use Encryption UDFs to encrypt and decrypt data in Open Source Saprk.

Syntax of Privacera Encryption UDFs for Apache Spark

The Privacera Crypto jar includes the following encryption-related UDFs.

Encrypt: With the quoted `'<encryption_scheme_name>'`, the `protect` UDF encrypts all values of `<column_name>` and writes the encrypted data to `<new_column_name>` in `<table_name>`:

```
select protect(<column_name>, <encryption_scheme_name>) as <new_column_name> from <datab
```

Decrypt: With the quoted `'<encryption_scheme_name>'`, the `unprotect` UDF decrypts all values of `<column_name>` and writes the decrypted data to `<new_column_name>` in `<table_name>`:

```
select unprotect(<column_name>, '<encryption_scheme_name>') as <new_column_name> from <d
```

Decrypt with obfuscation: With the quoted `'<encryption_scheme_name>'`, the `unprotect` UDF decrypts all values of `<column_name>`, further obfuscates the decrypted data via `<presentation_scheme_name>`, and writes the decrypted, obfuscated data to `<new_column_name>` in `<table_name>`:

```
select unprotect(<column_name>, '<encryption_scheme_name>', 'presentation_scheme_name')
```

For example usage, see Example queries to verify UDFs [52]

Download and install Privacera Crypto jar

To install the Privacera Crypto jar file in Apache Spark, get the URL of the Privacera Crypto jar file and download it to your Apache Spark, instance:

1. In your PrivaceraCloud account, go to **Settings** > **API Key**.
2. Under the **PEG** heading, for **PEG Crypto Starburst Trino Jar**, click **COPY URL**, and use that URL with `wget` on the command line of your Apache Spark instance. This URL is shown below as `<Privacera_Crypto_Jar_URL>`.

   ```
   cd <path_to_apache_spark_home_directory>
   wget <Privacera_Crypto_Jar_URL> -O privacera-crypto-jar-with-dependencies.jar
   ```
3. Copy the jar file to your Apache Spark instance's `plugins/privcera` directory, which you should create if it does not already exist.

Set up in Apache Spark

After you have downloaded the Privacera Crypto jar, you need to set some properties, update your Apache Spark start-up script, and define the UDFs.

Set variables in Apache Spark conf/crypto.properties

Create a file in Apache Spark called `<path_to_apache_spark_home_directory>/conf/crypto.properties`:

Add the following properties to the file, where:

- The value of your endpoint for Privacera Encryption on PrivaceraCloud, `<PrivaceraCloud_Encryption_URL>` is obtained by clicking the **Copy Url** link in **Settings** > **Api Key**

```
privacera.crypto.native.threadpool.size=100
privacera.crypto.shared.secret=secret
privacera.crypto.session.cache.size=1000
privacera.deployment.mode.saas=true
privacera.peg.base.url=<PrivaceraCloud_Encryption_URL>
privacera.peg.username=<PrivaceraCloud_Encryption_Username>
privacera.peg.password=<PrivaceraCloud_Encryption_Password>
```

Add envar to spark-env.sh

Follow these commands to define the path to the Privacera Crypto jar:

```
vi  <absolute_path_to_apache_spark_home_directory>/conf/spark-env.sh
export CRYPTO_CONFIG_DIR=<absolute_path_to_apache_spark_home_directory>/conf
```

Restart Apache Spark

```
# Go to Apache Spark bin directory
cd <absolute_path_to_trino_home_directory>/bin
# Restart Apache Spark
./spark-sql
```

Create Privacera protect and unprotect UDFs

To create both Privacera protect and unprotect user-defined function (UDF), run the following SQL commands inApache Spark :

```
create database if not EXISTS privacera;
drop function if exists privacera.protect;
drop function if exists privacera.unprotect;
drop function if exists privacera.mask;

CREATE FUNCTION privacera.protect AS 'com.privacera.crypto.PrivaceraEncryptUDF';
CREATE FUNCTION privacera.unprotect AS 'com.privacera.crypto.PrivaceraDecryptUDF';
CREATE FUNCTION privacera.mask AS 'com.privacera.crypto.PrivaceraMaskUDF';
```

Example queries to verify UDFs

See the syntax detailed in Syntax of Privacera Encryption UDFs for Apache Spark [51].

Encrypt: The following example query with the `protect` UDF encrypts the cleartext `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted`'EMAIL'` encryption scheme:

```
select protect(CUSTOMER_EMAIL, `EMAIL`) from CUSTOMERS;
```

Decrypt: The following example query with the `unprotect` UDF decrypts the encrypted `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted `'EMAIL'` encryption scheme:

```
select unprotect(CUSTOMER_EMAIL, 'EMAIL') from CUSTOMERS;
```

Decrypt with obfuscation: The following example query with the `unprotect` UDF decrypts the encrypted `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted `'EMAIL'` encryption scheme and obfuscates the decrypted data with the presentation scheme `PRESENTATION_EMAIL`:

```
select unprotect(CUSTOMER_EMAIL, 'EMAIL', 'PRESENTATION_EMAIL') as OPTIONAL_OUTPUT_COLUM
```

Hive UDFs for encryption on Privacera Platform

This topic provides instruction on how to enable encryption using a Hive user-defined function (UDF).

Add Privacera UDF in Hive

To add a Privacera user-defined function (UDF) in Hive, follow these steps:

1. Log in to the Privacera Portal.
2. From the navigation menu, select **Encryption & Masking** > **Encryption**.
3. Under the **Diagnostics** tab, click **Encryption**.
4. Click **Create for UDF - Protect, UnProtect**.
5. Click **Yes**.
   The UDF is created.

Confirm Privacera UDF in Hive

1. SSH to the instance.
2. Do kinit for `<user>`.
3. Connect to *beeline*.

   ```
   # Example
   beeline -u "jdbc:hive2://<hostname>:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNam
   ```
4. Enter the following two commands to describe the functions created by Privacera:
   - `DESCRIBE FUNCTION EXTENDED privacera.protect;`
   - `DESCRIBE FUNCTION EXTENDED privacera.unprotect;`

   NOTE
   If the UDF has not been created successfully, you will see the error message "**Function 'privacera.protect' does not exist**".

Test Privacera UDF in Hive

1. Log in to Privacera Portal.
2. From the navigation menu, select **Encryption & Masking** > **Schemes** .
3. Click **Add**.
4. Create a scheme by entering the following details:
   - Scheme Name: SWIFT_ID
   - Format type: FPE_ALPHA_NUMERIC
   - Scope: All
   - Algorithm: FPE
5. Click **Save**.
6. Check if the KMS key is generated for the scheme:
   a. Log in to Ranger at `http://<hostname>:6080` with the username **keyadmin**.
   b. Click **Encryption** > **Key Manager**.
7. SSH to the instance.
8. Do kinit for `<user>`.
9. Connect to beeline.

   ```
   #Example
   beeline -u "jdbc:hive2://<hostname>:2181/;
   serviceDiscoveryMode=zooKeeper;
   zooKeeperNamespace=hiveserver2" -n $<user>
   ```

10. After connecting to *beeline* enter the following commands.

    ```
    select privacera.protect("TEXT" ,"SWIFT_ID");
    select privacera.unprotect(privacera.protect("TEXT" ,"SWIFT_ID"), "SWIFT_ID");
    ```
11. Check the KMS audits in Ranger.
    - Log in to Ranger `http://<hostname>:6080` with the username **keyadmin**.
    - Click **Audit** > **Access**.
    - If the UDF fails, the message "denied access" is recorded in Ranger KMS audits. If access is denied, you need to give permission to `$<user>`.
    - If the UDF result is successful, the audits are shown as **Allowed**.
    - To check crypto UDF logs, run this command:

      ```
      sudo su
      tail -f /var/log/hive/hiveserver2.log
      ```

Give users access to UDFs

Users need access to the UDFs.

Set up Privacera Encryption

Check if the shaded JAR is needed

You might need to use a shaded JAR depending on certain conditions described below.

To check if the shaded JAR is needed, go to the Hive libraries directory:

```
cd /opt/cloudera/parcels/CDH/lib/hive/lib/
ls | grep http
```

You need to install the shaded JAR if:

- The version of the http client is less than 4.5.6.
- The UDF throws a NoClassDefFound exception for class `org/apache/http/config/Lookup`.

To install the shaded JAR, see Install the shaded JAR [55].

Create Privacera UDF in CDH/CDP

To install the Privacera Encryption JAR in CDH/CDP cluster, do the following:

1. Download the JAR to the cluster node.

   ```
   sudo su - privacera
   exportPRIVACERA_BASE_DOWNLOAD_URL=$<PRIVACERA_BASE_DOWNLOAD_URL>
   wget $<PRIVACERA_BASE_DOWNLOAD_URL>/privacera-crypto-jar-with-dependencies.jar -O pri
   ```
2. Upload the JAR into an HDFS location where Hive can access it. The following are the commands to upload the JAR into HDFS using Hadoop CLI:

   ```
   kinit -kt /opt/privacera/keytab/privacera.headless.keytab privacera
   hadoop fs -ls
   hadoop fs -mkdir -p /privacera/crypto/jars
   hadoop fs -put privacera-crypto-jar-with-dependencies.jar /privacera/crypto/jars/priv
   ```
3. Create configuration files.

   ```
   hadoop fs -mkdir -p /privacera/crypto/configs
   ```

   ```
   vi crypto.properties
   ```

   ```
   privacera.portal.base.url=http://$<PRIVACERA_SERVER>:6868
   privacera.portal.username=$<USER_NAME>
   ```

```
privacera.portal.password=$<PASSWORD>
#Mode of encryption/decryption rpc/native
privacera.crypto.mode=native

cp crypto.properties  crypto_default.properties
```

4. Upload the configuration files to DBFS.

```
hadoop fs -put crypto.properties /privacera/crypto/configs/crypto.properties
hadoop fs -put crypto_default.properties /privacera/crypto/configs/crypto_default.pro
```

5. Install the Crypto JAR into the Cluster.
   a. SSH to the cluster node.
   b. Do kinit for `$<user>` (if Kerberos is enabled).
   c. Connect to beeline.
   d. If you have Kerberos-enabled cluster, use the following command to login to beeline:
      Update the values for for the variables `<HIVE_SERVER2_HOSTNAME>` and `<REALM>` below.

      ```
      HIVE_SERVER2_HOSTNAME=<<10.1.1.2>>REALM=<<EXAMPLE.PRIVACERA.US>>beeline -u "jdbc:
      ```

6. Add Privacera crypto UDF JAR. You need to run multiple SQL queries in Databricks cluster to create privacera encryption functions.
   - SQL query to create the Privacera unprotect UDF.

     ```
     add jar hdfs:///privacera/crypto/jars/privacera-crypto-jar-with-dependencies.jar;
     ```

7. Create the Privacera encryption UDF. You need to run multiple SQL queries in beeline to create Privacera encryption UDF.
   - SQL query to create Privacera unprotect UDF:

     ```
     create database if not exists privacera ;
     drop functionif exists privacera.unprotect;
     create function privacera.unprotect AS 'com.privacera.crypto.PrivaceraDecryptUDF' u
     ```

   - SQL query to create Privacera protect UDF:

     ```
     drop functionif exists privacera.protect;
     create function privacera.protect AS 'com.privacera.crypto.PrivaceraEncryptUDF' usi
     ```

Install the shaded JAR

To install the shaded Privacera Encryption JAR in a CDH/CDP cluster, follow these steps:

1. Download JAR in cluster node.

   ```
   sudo su - privacera
   exportPRIVACERA_BASE_DOWNLOAD_URL=$<PRIVACERA_BASE_DOWNLOAD_URL>
   wget $<PRIVACERA_BASE_DOWNLOAD_URL>/privacera-crypto-jar-with-dependencies-shaded.jar
   ```

2. Upload JAR into HDFS location from where Hive can access it. Following are the commands to upload JAR into HDFS using Hadoop CLI.

   ```
   kinit -kt /opt/privacera/keytab/privacera.headless.keytab privacera
   hadoop fs -ls /
   hadoop fs -mkdir -p /privacera/crypto/jars
   hadoop fs -put privacera-crypto-jar-with-dependencies-shaded.jar /privacera/crypto/ja
   ```

   3. Create configuration files.

   ```
   hadoop fs -mkdir -p /privacera/crypto/configs
   vi crypto.properties
   privacera.portal.base.url=http://$<PRIVACERA_SERVER>:6868
   privacera.portal.username=$<USER_NAME>
   privacera.portal.password=$<PASSWORD>
   ```

```
#Mode of encryption/decryption rpc/native
privacera.crypto.mode=native
cp crypto.properties  crypto_default.properties
```

3. Upload the configuration files to DBFS.

   ```
   hadoop fs -put crypto.properties /privacera/crypto/configs/crypto.properties
   hadoop fs -put crypto_default.properties /privacera/crypto/configs/crypto_default.pro
   ```

4. Install Crypto JAR into cluster.
   a. SSH to the cluster node.
   b. Do kinit for `<user>` (if Kerberos is enabled).
   c. Connect to beeline.
   d. If you have Kerberos-enabled cluster, use the following command to login to beeline:
      Update the values for `<HIVE_SERVER2_HOSTNAME>` and `<REALM>` as per your environment.

      ```
      HIVE_SERVER2_HOSTNAME=<<10.1.1.2>>REALM=<<EXAMPLE.PRIVACERA.US>>beeline -u "jdbc:h
      ```

5. Add Privacera Crypto UDF JAR. You need to run multiple SQL queries in Databricks cluster to create Privacera encryption functions.
   - SQL query to create the Privacera unprotect UDF.

     ```
     add jar hdfs:///privacera/crypto//privacera-crypto-jar-with-dependencies-shaded.jar
     ```

6. Create the Privacera encryption UDF. You need to run multiple SQL queries in beeline to create Privacera encryption UDF.
   - SQL query to create Privacera unprotect UDF .

     ```
     create database if not exists privacera ;
     drop functionif exists privacera.unprotect;
     CREATE FUNCTION privacera.unprotect AS 'com.privacera.crypto.PrivaceraDecryptUDF' u
     ```

   - SQL query to create Privacera protect UDF.

     ```
     drop functionif exists privacera.protect;
     CREATE FUNCTION privacera.protect AS 'com.privacera.crypto.PrivaceraEncryptUDF' usi
     ```

Hive TEZ: Add properties file in beeline

If you are using Hive TEZ, *before executing the UDFs*, you must add the cryptop.properties file in beeline with the following command:

```
add file hdfs:///privacera/crypto/configs/crypto.properties;
```

Sample queries to verify setup

- Sample query to run encryption:

  ```
  select privacera.protect("test_data","$<SCHEME_NAME>") limit 10;
  ```

- Sample query to run encryption and decryption in one query to verify setup:

  ```
  select privacera.unprotect(privacera.protect("test_data","$<SCHEME_NAME>"),"$<SCHEME_NA
  ```

- For authorization and leveraging Hive Masking policies, you need to install the Hive plug-in in hive-server2.

If you do not want to install the Hive plug-in, you can authorize use of the keys based on KMS.

- Create a view on top of your raw table:

  ```
  create view secure_view as select col1, privacera.protect(col2, ‘SSN’) as col2 from db
  select * from secure_view;
  ```

- If the user is not present in Privacera, they can still access the protect function. It is recommended to use the Hive plug-in as they can control the access to the resource using Ranger Policies and it makes it easier to manage them with the simple UI.

StreamSets Data Collector (SDC) and Privacera Encryption on Privacera Platform

This topic provides instruction on how to install and configure the Privacera StreamSets plugin for Ranger and Privacera Encryption.

Enable Encryption for SDC

To enable Privacera Encryption for the StreamSets Data Collector (SDC), do the following:

1. Run the following command:

   ```
   cd ~/privacera/privacera-manager/config
   cp sample-vars/vars.crypto.streamset.yml custom-vars/vars.crypto.streamset.yml
   ```
2. Update Privacera Manager:

   ```
   cd ~/privacera/privacera-manager/
   ./privacera-manager.sh update
   ```

Configure Encryption for SDC

1. Copy the StreamSets Privacera package.
   a. If you have StreamSets and Privacera Manager running on different systems, copy the following two files from `~/privacera/privacera-manager/output/streamset/` on the Privacera Manager host machine:
      - privacera-streamset.tar.gz
      - crypto-config

      If you have JCEKS enabled, copy the following file from the location, `~/privacera/privacera-manager/config/keystores/` of the Privacera Manager host machine:
      - cryptoprop.jceks

   b. If you have StreamSets and Privacera Manager running on same system, do the following:

      ```
      cp ~/privacera/privacera-manager/output/streamset/privacera-streamset.tar.gz ~/pr
      cp -r ~/privacera/privacera-manager/output/streamset/crypto-config ~/privacera/do
      ```

      If you have JCEKS enabled, do the following:

      ```
      cp ~/privacera/privacera-manager/config/keystores/cryptoprop.jceks ~/privacera/do
      ```
2. Extract the StreamSets Privacera package.

   ```
   cd ~/privacera/downloads
   mkdir streamsets
   tar xfz ~/privacera/downloads/privacera-streamset.tar.gz -C streamsets
   ```
3. Access the StreamSets installation directory as root user.

   ```
   sudo su
   ```
4. Set the StreamSets installation directory.

   ```
   export STREAMSET_HOME=/opt/streamset/streamsets-datacollector-3.13.0
   ```
5. Copy the Privacera library into the StreamSets data collector `user-libs` directory:

   ```
   cp -r streamsets/privacera-streamset/ $<STREAMSET_HOME>/user-libs/
   ```
6. Copy the configuration files.

   ```
   cp -r crypto-config $<STREAMSET_HOME>/../crypto-config
   ```

7. Define a security policy.

   ```
   cat << EOF >> $<STREAMSET_HOME>/etc/sdc-security.policy
   grant <
   permission java.io.FilePermission "/opt/privacera/-", "read";
   permission java.io.FilePermission "/opt/streamset/-", "read,write";
   permission java.net.SocketPermission "*", "connect,accept,listen,resolve";
   >;
   EOF
   ```
8. Stop StreamSets.

   ```
   kill -9 $(ps aux | grep 'sdc'| awk '<print $2>')
   ```
9. Restart StreamSets.

   ```
   ulimit -n 32768
   nohup $<STREAMSET_HOME>/bin/streamsets dc &
   ```
10. Verify the logs to make sure that StreamSets is running.

    ```
    tail -f $<STREAMSET_HOME>/log/sdc.log
    ```

Verify StreamSets setup

To verify that Privacera Encryption is now working with the StreamSets Data Collector (SDC), follow these steps:

1. Configure a sample pipeline to encrypt a local file. You can use the following sample. Import this sample pipeline into StreamSets. For more information, see Sample pipeline.
2. Access the StreamSets installation directory as root user.

   ```
   sudo su
   ```
3. Create data directories.

   ```
   DATA_DIR=/opt/streamset/
   cd $<DATA_DIR>
   mkdir -p customer_data/input
   mkdir -p customer_data/output
   mkdir -p customer_data/input_error
   mkdir -p customer_data/output/encrypted_error
   ```
4. Create a sample data file:

   ```
   cat << EOF > customer_data/input/customer_data_with_header.csv
   id,name,ssn,email_address,amount
   1,Tamara,898453744,aphillips@vang.info,162454.67
   2,Richard,65511350,vreynolds@gmail.com,602.89
   3,Tanya,634090950,harringtonwilliam@diaz-king.com,48712.67
   4,Richard,829439881,martinvalerie@yahoo.com,5122.02
   5,Raymond,227804351,sarachavez@yahoo.com,97963.857
   6,Melissa,553465892,kevinwillis@gmail.com,36654.806
   7,Deborah,782539839,brittney24@yahoo.com,19.231
   8,Rodney,515337130,jenniferkelly@davis-bond.biz,65083.651
   9,Katherine,137057143,jperkins@gmail.com,4822.343
   10,David,432941241,wmccann@hotmail.com,4069.34
   EOF
   ```
5. Create a metadata file to map the input dataset columns to Privacera Encryption schema columns:

   ```
   cat << EOF > customer_data/customer_data.meta
   COLUMN_NAME|SCHEME_NAME
   id|
   ```

```
name|SYSTEM_PERSON_NAME
ssn|SYSTEM_SSN
email_address|SYSTEM_EMAIL
amount|
EOF
```

To run the sample pipeline, make sure you have the Privacera user created in your Ranger and it has permissions on the KMS keys starting with *pmsk**.

Add permission for keys in Ranger

1. Log in to the Ranger UI as an administrator and create the Privacera user. You can grant permissions to the Privacera user on keys.
2. Log in to Ranger with keyadmin credentials and click on `privacera_kms`.
3. Create or update policy for Privacera user.
4. Run the StreamSets pipeline preview and verify the encrypted value on the right side of the table.

Trino UDFs for encryption and masking on Privacera Platform

This topic provides instruction on how to install and configure the Privacera crypto plugin for Trino. Doing so will allow you to use Privacera-supplied encryption user-defined functions (UDFs) in Trino to encrypt or decrypt data.

The protect and unprotect UDFs work with `privacera_starburstenterprise` but not with `privacera_hive`. Starburst has three possible configurations (Hive, System, and Hive + System), of which only the system-level has been verified.

Privacera Encryption UDFs for Trino

The Privacera crypto plugin includes the following UDFs:

- **Encrypt**: With the quoted `<encryption_scheme_name>`, the `protect` UDF encrypts all values of `<column_name>` in a table:

  ```
  select protect(<column_name>, '<encryption_scheme_name>') from <table_name>;
  ```
- **Decrypt**: With the `<encryption_scheme_name>`, the `unprotect` UDF decrypts all values of `<column_name>` in a table:

  ```
  select unprotect(<column_name>, '<encryption_scheme_name>') from <table_name>;
  ```
- **Decrypt with obfuscation**: With the quoted `<encryption_scheme_name>`, the `unprotect` UDF decrypts all values of `<column_name>` in a table, further obfuscates the decrypted data via `<presentation_scheme_name>`, and writes the decrypted, obfuscated data to `<optional_column_name_for_obfuscated_data>`:

  ```
  select unprotect(<column_name>, '<encryption_scheme_name>', <presentation_scheme_name>
  ```
- **Decrypt with obfuscation**: With the quoted `<encryption_scheme_name>`, the `unprotect` UDF decrypts all values of `<column_name>` in a table, further obfuscates the decrypted data via `<presentation_scheme_name>`, and writes the decrypted, obfuscated data to `<optional_column_name_for_obfuscated_data>`:

  ```
  select unprotect(<column_name>, '<encryption_scheme_name>', <presentation_scheme_name>
  ```

  For example usage, see Example Queries to Verify Privacera-supplied UDFs [62].

Prerequisites for installing Privacera crypto plugin for Trino

Before installing the Privacera crypto plugin for Trino, do the following:

- Install Trino. In this topic, the location of the installed Trino software is shown as:

`<absolute_path_to_trino_home_directory>`

- Identify the users who will use the UDFs and ensure they have access to the pertinent tables.
- Determine the required paths to the crypto JAR and `crypto.properties file`. The Encryption plugin for Trino relies on these files. The paths for each file depend on whether you have deployed Trino in a container (such as Docker). These different paths are detailed in the following sections.

Install the Privacera crypto plugin for Trino using Privacera Manager

To install the Privacera crypto plugin, follow these steps:

1. Upgrade Privacera Manager to get a shell script. [60]
2. Configure the Privacera crypto plugin for Trino. [60]
3. Run the shell script to install the Privacera crypto plugin [60].
4. Verify that the shell script ran correctly.
5. Restart Trino to register the Privacera crypto UDFs for Trino [61].

See the following sections for details about how to complete each step.

Upgrade Privacera Manager

To install the Privacera crypto plugin, you first need to update Privacera Manager to get a shell script. This shell script downloads the Privacera Encryption crypto plugin for Trino.

To do so, run the following commands:

```
# Change to Privacera Manager directory
cd ~/privacera/privacera-manager

# Upgrade Privacera Manager itself
 ./privacera-manager.sh upgrade-manager
```

Configure the Privacera crypto plugin for Trino

```
# Copy the Trino properties file to Privacera Manager config/custom-vars directory
cp config/sample-vars/vars.starburst.enterprise.trino.yml config/custom-vars/

# Set property STARBURST_TRINO_ENABLE to true
vi config/custom-vars/vars.starburst.enterprise.trino.yml
...
STARBURST_TRINO_ENABLE: "true"
...
# Save the file
# Edit starburst-trino-crypto.yml to specify Trino home directory
vi ansible/privacera-docker/roles/defaults/main/starburst-trino-crypto.yml
...
STARBURST_TRINO_INSTALL_DIR: <absolute_path_to_trino_home_directory>
...
# Save the file
```

Run shell script to install Privacera crypto plugin

```
# Change to Privacera Manager directory
cd ~/privacera/privacera-manager

# Update Privacera Manager to get shell script
./privacera-manager.sh update

# Change to new directory created by privacera-manager update
```

```
cd output/starburst-trino-crypto/

# Make the script executable
chmod +x privacera_crypto_trino_setup.sh
#
######################################
# NOTE: You must copy the script to your Trino or Starburst instance
######################################
#
#  Run the script on your instance from where you copied it
./privacera_crypto_trino_setup.sh
```

Verify that the shell script ran correctly

Verify the following:

- The location of the Privacera crypto JAR:

  ```
  # For non-container deployment
  ls -l <absolute_path_to_trino_home_directory>/plugin/privacera/privacera-crypto-jar-wi

  # For container deployment
  ls -l /data/starburst/plugin/privacera/privacera-crypto-jar-with-dependencies.jar
  ```
- The location of the `crypto.properties` file in Trino's `etc` directory:

  ```
  # Verify existence of crypto.properties file
  # For non-container deployment
  ls -l <absolute_path_to_trino_home_directory>/etc/crypto.properties

  # For non-container deployment
  ls -l /data/starburst/etc/crypto.properties
  ```

Restart Trino to register the Privacera crypto UDFs for Trino

```
# Go to Trino bin directory
cd /<trino_installation_directory>/bin

# Restart Trino
./launcher restart
```

privacera.unprotect with optional presentation scheme

The `unprotect` UDF supports an optional specification of a presentation scheme that further obfuscates the decrypted data.

Syntax:

```
select <id>, privacera.unprotect(<COLUMN_NAME>, <ENCRYPTION_SCHEME_NAME>, <PRESENTATION_
```

where:

- `<PRESENTATION_SCHEME_NAME>` is the name of the chosen Privacera presentation scheme with which to further obfuscate the decrypted data.
- `<OPTIONAL_NAME_FOR_COLUMN_TO_WRITE_OBFUSCATED_OUTPUT>` is a "pretty" name for the column that the obfuscated data is written to.
- Other arguments are the same as in the preceding `unprotect` example.

Example queries to verify Privacera-supplied UDFs

See the syntax detailed in Syntax of Privacera Encryption UDFs for Trino [59].

- **Encrypt**: The following example query with the `protect` UDF encrypts the cleartext `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted`'EMAIL'` encryption scheme:

  ```
  select protect(CUSTOMER_EMAIL, `EMAIL`) from CUSTOMERS;
  ```
- **Decrypt**: The following example query with the `unprotect` UDF decrypts the encrypted `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted `'EMAIL'` encryption scheme:

  ```
  select unprotect(CUSTOMER_EMAIL, 'EMAIL') from CUSTOMERS;
  ```
- **Decrypt with obfuscation**: The following example query with the `unprotect` UDF decrypts the encrypted `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted `'EMAIL'` encryption scheme, obfuscates the decrypted data with the presentation scheme `PRESENTATION_EMAIL`, and writes the decrypted, obfuscated data to `OPTIONAL_OUTPUT_COLUMN_FOR_OBFUSCATED_DATA`:

  ```
  select unprotect(CUSTOMER_EMAIL, 'EMAIL', PRESENTATION_EMAIL) OPTIONAL_OUTPUT_COLUMN_F
  ```

Privacera Encryption UDFs for Starburst Enterprise Trino on PrivaceraCloud

This section describes how to install and configure the Privacera jar in Trino in order to use the Privacera-supplied Encryption UDFs to encrypt and decrypt data in Trino.

These encryption UDFs are defined in the Privacera Crypto jar. You don't have to define them.

NOTE

The protect and unprotect UDFs work properly with `privacera_starburstenterprise` but not with `privacera_hive`. Starburst has three possible configurations (Hive, System, and Hive + System), of which only the system-level has been verified.

Syntax of Privacera Encryption UDFs for Trino

The Privacera Crypto jar includes the following encryption-related UDFs. The Privacera Crypto jar also includes a `mask` UDF. See Privacera Encryption UDF for masking in Trino on PrivaceraCloud [64].

Encrypt: With the quoted `'<encryption_scheme_name>'`, the `protect` UDF encrypts all values of `<column_name>` in `<table_name>`:

```
select protect(<column_name>, <encryption_scheme_name>) from <table_name>;
```

Decrypt: With the quoted `'<encryption_scheme_name>`', the `unprotect` UDF decrypts all values of `<column_name>` in `<table_name>`:

```
select unprotect(<column_name>, '<encryption_scheme_name>') from <table_name>;
```

Decrypt with obfuscation: With the quoted `'<encryption_scheme_name>`', the `unprotect` UDF decrypts all values of `<column_name>`, further obfuscates the decrypted data via `<presentation_scheme_name>`, and and writes the decrypted, obfuscated data to `<optional_column_name_for_obfuscated_data>` in `<table_name>`:

```
select unprotect(<column_name>, '<encryption_scheme_name>' <optional_column_name_for_obf
```

For example usage, see Example Queries to Verify Privacera-supplied UDFs [63].

Prerequisites for installing Privacera Crypto plug-in for Trino

The following should already be ready:

- A fully functional installation of Trino. In these examples, the location of the installed Trino software is shown as `<absolute_path_to_trino_home_directory>`.
- The users who will use the UDFs have sufficient access to the pertinent tables.

Download and install Privacera Crypto jar

To install the Privacera Crypto jar file in Trino:

1. In your PrivaceraCloud account, go to **Settings > Applications > Trino** or **Starburst Enterprise**.
2. Choose **CONNECT NEW APPLICATION** or **Edit Already Existing Application**.
3. Choose **Encryption / Decryption**. Under **PEG Crypto Starburst Trino Jar**, click **DOWNLOAD JAR**.
   Alternatively, click **COPY URL** and use that URL with wget on the command line of your Trino instance.
4. Save the jar file.
5. Copy the jar file to your Trino instance `plugins/privcera` directory. If the directory doesn't exist, create it.

Set variables in Trino etc/crypto.properties

Create a file in Trino called `etc/crypto.properties` in one of the following locations:

- For non-container deployment: `<absolute_path_to_trino_home_directory>/etc/crypto.properties`.
- For container deployment: `/data/starburst/etc/crypto.properties`.

Add the following properties to the file, where:

- `<PrivaceraCloud_Encryption_URL>` is obtained by clicking the **Copy Url** link in **Settings** > **Api Key**
- `<PrivaceraCloud_Encryption_Username>` and `<PrivaceraCloud_Encryption_Password>` are obtained from **Settings** > **Account**. Under the **PRIVACERA ENCRYPTION** heading, click **Edit** to display the **Privacera Encryption Configuration** popup window with the username and password.

```
privacera.crypto.native.threadpool.size=100
privacera.crypto.shared.secret=secret
privacera.crypto.session.cache.size=1000
privacera.deployment.mode.saas=true
privacera.peg.base.url=<PrivaceraCloud_Encryption_URL>
privacera.peg.username=<PrivaceraCloud_Encryption_Username>
privacera.peg.password=<PrivaceraCloud_Encryption_Password>
```

Restart Trino to register the Privacera encryption and masking UDFs for Trino

```
# Go to Trino bin directory
cd <absolute_path_to_trino_home_directory>/bin
# Restart Trino
./launcher restart
```

Example queries to verify Privacera-supplied UDFs

See the syntax detailed in Syntax of Privacera Encryption UDFs for Trino [62].

Encrypt: The following example query with the `protect` UDF encrypts the cleartext `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted`'EMAIL'` encryption scheme:

```
select protect(CUSTOMER_EMAIL, `EMAIL`) from CUSTOMERS;
```

Decrypt: The following example query with the `unprotect` UDF decrypts the encrypted `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted `'EMAIL'` encryption scheme:

```
select unprotect(CUSTOMER_EMAIL, 'EMAIL') from CUSTOMERS;
```

Decrypt with obfuscation: The following example query with the `unprotect` UDF decrypts the encrypted `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted `'EMAIL'` encryption scheme:

```
select unprotect(CUSTOMER_EMAIL, 'EMAIL' PRESENTATION_EMAIL) OPTIONAL_OUTPUT_COLUMN_FOR_
```

Privacera Encryption UDF for masking in Trino on PrivaceraCloud

Privacera Encryption includes a UDF for Trino that can one-way mask your data. For background, see Masking schemes [44].

Syntax of Trino UDF for masking

The masking UDF for Databricks has the following syntax:

Mask: With the quoted `'<mask_scheme_name>'`, the `mask` UDF one-way transforms all values of `<column_name>` in `<table_name>`:

```
select mask(<column_name>, <mask_scheme_name>) from <table_name>;
```

Prerequisites for Trino masking UDF

The following should already be ready:

- A fully functional installation of Trino.
- The Privacera init script for Trino must be installed in your Trino instance. See Connect Trino to PrivaceraCloud.
- The Privacera Crypto jar, which includes the mask UDFs, must be installed in your Trino instance. See Download and install Privacera Crypto jar [63]Trino UDFs for encryption and masking on Privacera Platform [59].
- The users who will use the UDFs have sufficient access to the pertinent tables in Trino.

Mask UDF pre-defined in Trino

The `mask` UDF comes pre-defined in the Privacera Crypto jar. You do not need to define it yourself.

Example query to verify Privacera-supplied mask UDF

See the syntax detailed in Syntax of Trino UDF for masking [64]

Mask: The following example query with the `mask` UDF one-way transforms the cleartext `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted`'MASK_SCHEME_EMAIL'` masking scheme:

```
select mask(CUSTOMER_EMAIL, `MASK_SCHEME_EMAIL`) from CUSTOMERS;
```

Redact the column `email` from the `customer_data` database with the masking scheme `EMAIL_REDACT_SCHEME` and save the output to a column called `RedactedEmail`.

```
select mask(email,'EMAIL_REDACT_SCHEME')
as RedactedEmail
db.customer_data;
```

Single query to encrypt and mask: Encrypt (`protect`) the column `PERSON_NAM` from the `customer_data` database with the `PERSON_NAME_ENCRYPTION_SCHEME` and mask the `EMAIL` from the `customer_data` database with the masking scheme `EMAIL_MASKING_SCHEME`. The data are transformed in place with no intermediate location.

```
select protect(PERSON_NAME,'PERSON_NAME_ENCRYPTION_SCHEME'),
mask(EMAIL,'EMAIL_MASKING_SCHEME')
from db.customer_data;
```

Databricks UDFs for Encryption

You can create Privacera Encryption user-defined functions (UDFs) by running SQL queries in your Databricks cluster.

Create Privacera protect UDF

To create a Privacera protect user-defined function (UDF), run the following SQL query in your Databricks cluster:

```
create database if not exists privacera;
drop function if exists privacera.protect;
CREATE FUNCTION privacera.protect AS 'com.privacera.crypto.PrivaceraEncryptUDF';
```

Create Privacera unprotect UDF

To create a Privacera unprotect user-defined function (UDF), run the following SQL query in your Databricks cluster:

```
create database if not exists privacera;
drop function if exists privacera.unprotect;
CREATE FUNCTION privacera.unprotect AS 'com.privacera.crypto.PrivaceraDecryptUDF';
```

Run sample queries in Databricks to verify

You can run queries to verify that the definitions of the UDFs in Databricks actually do work. These queries do the following:

1. Run the `protect` UDF to encrypt a database column shown as a variable `<colname>` . Substitute your own column name for this variable.
2. Run the `unprotect` UDF to decrypt that same database column.

Sample query to run encryption:

```
select privacera.protect($<colname>,'$<SCHEME_NAME>') from $<db_name>.$<table_name> limi
```

Sample query to run encryption and decryption in a single query to verify the setup:

```
select privacera.unprotect(privacera.protect($<colname>,'$<SCHEME_NAME>'),'$<SCHEME_NAME
```

Create a custom path to the crypto properties file in Databricks

For your Databricks UDFs, you might want to change the location of Privacera's crypto properties file in your Databricks cluster to enhance security. The `crypto.properties` file contains configuration settings for Privacera Encryption.

To change the location of the `crypto.properties` file, follow these steps:

1. Move the properties file to a new directory on your Databricks cluster.
2. Define an environment variable in Databricks to point to that new directory.
3. Define the same path in a Privacera custom variable on your Privacera host.

The following sections describe how to complete each of the above steps.

NOTE
This is to change the location of the crypto properties file on your Databricks cluster, *not* a DBFS location.

Move the crypto properties file to new location in Databricks

In your Databricks cluster, the default location of the Privacera crypto properties file is `/databricks/crypto/config/crypto.properties`. This an absolute path starting with `/`.

1. On your Privacera host, move the properties file from its default location to the new path. This must be an absolute path starting with `/`.
2. Make a note of this new path.

In the steps here, this new location is called `<absolute_path_on_databricks_cluster_to_directory_with_crypto.properties_file>`.

Define an environment variable in Databricks

You must set an environment variable to point to the new location of the Privacera crypto properties file.

As the Databricks administrator, in your Databricks cluster, do the following:

1. Navigate to the system **Configuration** tab.
2. In the **Environment Variables** section, add the following line: `CRYPTO_CONFIG_DIR=<absolute_path_on_databricks_cluster_to_directory_with_crypto.properties_file>`.
3. Save the change.

Define custom variable in Privacera

You need to define the same new path of the crypto properties file in your Privacera installation and update the configuration.

As the Privacera administrator, on the Privacera host, run the following commands:

```
cd ~/privacera/privacera-manager
cp config/sample-vars/vars.databricks.plugin.yml  config/custom-vars/vars.databricks.plu
vi config/custom-vars/vars.databricks.plugin.yml
DATABRICKS_CRYPTO_CONFIG_DIR: "<absolute_path_on_databricks_cluster_to_directory_with_cr
# Save the file
# Update the configuration
cd ~/privacera/privacera-manager
./privacera-manager.sh update
```

Create and run Databricks UDF for masking

You can create Privacera user-defined functions (UDFs) for masking by running the following SQL query in your Databricks cluster:

```
drop function if exists db.mask;
CREATE FUNCTION db.mask AS 'com.privacera.crypto.PrivaceraMaskUDF'
```

Run sample queries to verify masking and encryption

Redact the column `email` from the `customer_data` database with the masking scheme `EMAIL_REDACT_SCHEME` and save the output to a column called `RedactedEmail`.

```
select mask(email,'EMAIL_REDACT_SCHEME')
as RedactedEmail
db.customer_data;
```

Single query to encrypt and mask: Encrypt (`protect`) the column `PERSON_NAM` from the `customer_data` database with the `PERSON_NAME_ENCRYPTION_SCHEME` and mask the `EMAIL` from the `customer_data` database with the masking scheme `EMAIL_MASKING_SCHEME`. The data are transformed in place with no intermediate location.

```
select protect(PERSON_NAME,'PERSON_NAME_ENCRYPTION_SCHEME'),
mask(EMAIL,'EMAIL_MASKING_SCHEME')
from db.customer_data;
```

Privacera Encryption UDF for masking in Databricks on PrivaceraCloud

Privacera Encryption includes a UDF for Databricks that can one-way mask your data. For background, see Masking schemes [44].

Syntax of Databricks UDF for masking

The masking UDF for Databricks has the following syntax:

Mask: With the quoted `'<mask_scheme_name>'`, the `mask` UDF one-way transforms all values of `<column_name>` in `<table_name>`:

```
select mask(<column_name>, <mask_scheme_name>) from <table_name>;
```

Prerequisites for Databricks masking UDF

The following should already be ready:

- The Privacera init script for Databricks must be installed in your Databricks instance. See Connect Databricks to PrivaceraCloud.
- A fully functional installation of Databricks.
- The users who will use the UDFs have sufficient access to the pertinent tables in Databricks.

Define the mask UDF in Databricks

In your Databricks instance, run the following command to define the `mask` UDF:

```
drop function if exists db.mask;
CREATE FUNCTION db.mask AS 'com.privacera.crypto.PrivaceraMaskUDF'
```

Example query to verify Privacera-supplied mask UDF

See the syntax detailed in Syntax of Databricks UDF for masking [67].

Mask: The following example query with the `mask` UDF one-way transforms the cleartext `CUSTOMER_EMAIL` column of the `CUSTOMERS` table using the quoted`'MASK_SCHEME_EMAIL'` masking scheme:

```
select mask(CUSTOMER_EMAIL, `MASK_SCHEME_EMAIL`) from CUSTOMERS;
```

Redact the column `email` from the `customer_data` database with the masking scheme `EMAIL_REDACT_SCHEME` and save the output to a column called `RedactedEmail`.

```
select mask(email,'EMAIL_REDACT_SCHEME')
as RedactedEmail
db.customer_data;
```

Single query to encrypt and mask: Encrypt (`protect`) the column `PERSON_NAM` from the `customer_data` database with the `PERSON_NAME_ENCRYPTION_SCHEME` and mask the `EMAIL` from the `customer_data` database with the masking scheme `EMAIL_MASKING_SCHEME`. The data are transformed in place with no intermediate location.

```
select protect(PERSON_NAME,'PERSON_NAME_ENCRYPTION_SCHEME'),
mask(EMAIL,'EMAIL_MASKING_SCHEME')
from db.customer_data;
```

Set up Databricks encryption and masking

This topic describes how to install and configure the Privacera Encryption JAR file UDF in Privacera Manager Databricks to create UDFs for encryption and masking and to create policies for users and groups.

The overall approach is as follows:

1. Install the Privacera Manager Encryption JAR in Databricks with the Databricks CLI or UI
2. Upload Privacera Manager configuration files to Databricks
3. Define UDFs in Databricks to call the Privacera Manager encryption `protect` and `unprotect` methods.

Prerequisites

- In Databricks, make sure that the users who will use the UDFs have sufficient access to write the pertinent tables.
- In Privacera Manager, make sure to configure the Databricks datasource: Databricks Spark Plugin (Python/SQL) on AWS, Azure, or GCP.
- In Privacera Manager, make sure that Privacera Encryption has been enabled. .
- In Privacera Manager, make sure that the users who will use the UDFs in Databricks have been given permission to access the Create scheme policies [46] that are part of the UDF syntax.
- In Privacera Manager, make sure that these same users have been given permission to Provide user access to Ranger KMS [14].

Methods for installing the Privacera encryption JAR

You can install the Privacera encryption JAR file in the following ways:

- Using the Databricks command-line interface [68]
- Using the Databricks web-based user interface [69]

After you install the JAR file, you need to define some configuration properties and User-Defined Functions (UDFs) to call the Privacera encryption `/protect` and `/unprotect` API endpoints.

Install the Privacera encryption JAR using Databricks CLI

To install the Privacera encryption JAR using the Databricks CLI, follow these steps:

1. Download the JAR to a local machine.
   The variable `PRIVACERA_BASE_DOWNLOAD_URL` depends on the version of the Privacera software you want. See Install Privacera Manager on Privacera Platform.

   ```
   export PRIVACERA_BASE_DOWNLOAD_URL=$<PRIVACERA_BASE_DOWNLOAD_URL>
   wget $<PRIVACERA_BASE_DOWNLOAD_URL>/privacera-crypto-jar-with-dependencies.jar -O pri
   ```
2. Upload the JAR file to DBFS or an S3 location from where the Databricks cluster can access it.
3. Upload the jar into DBFS using the Databricks CLI:

   ```
   databricks fs ls
   databricks fs mkdirs dbfs:/privacera/crypto/jars
   databricks fs cp privacera-crypto-jar-with-dependencies.jar dbfs:/privacera/crypto/ja
   ```

Install the Privacera encryption JAR using Databricks UI

To install the Privacera encryption JAR using the Databricks UI, follow these steps:

1. Navigate to the Databricks cluster details page by selecting **Clusters** > cluster name > **Libraries**.
2. Click **Install** > **New**.
3. Drop or upload the JAR file.
   `dbfs:/privacera/crypto/jars/privacera-crypto-jar-with-dependencies.jar`
4. Wait until the JAR file is installed.

Create and upload encryption configuration files

The steps here rely on the default location of the Privacera crypto properties file. However, you can change this location to a directory of your choice. Follow the steps here and then see Create a custom path to the crypto properties file in Databricks [65].

To create and upload the encryption configuration files, do the following:

1. Create the configuration file on your local machine. In the next step, upload the file to the Databricks cluster.

   ```
   mkdir -p privacera/crypto/configs
   cd privacera/crypto/configs
    # Edit the crypto_default.properties file to set the following variables.
   vi crypto_default.properties
   privacera.portal.base.url=http://<APP_HOSTNAME.>:6868
   privacera.portal.username=<SOME_USERNAME>
   privacera.portal.password=<SOME_PASSWORD>
    # Mode of encryption/decryption: rpc or native
   privacera.crypto.mode=native
   ```
2. Upload the configuration file to DBFS.

   ```
   databricks fs ls
   databricks fs mkdirs dbfs:/privacera/crypto/configs
   databricks fs cp crypto_default.properties dbfs:/privacera/crypto/configs/crypto_defa
   ```