Fortanix®

API Guide

FORTANIX DSM USE CASES

Fortanix Inc. | 800 West El Camino Real | Suit 180 | Mountain View, CA 94040 | United States of America | ☎ +1 (650) 943-2484 |
✉info@fortanix.com | 🌐 www.fortanix.com

TABLE OF CONTENTS

1.0 OVERVIEW

This document describes the Fortanix Data Security Manager (DSM) use cases for REST API.

2.0 FORTANIX DATA SECURITY MANAGER USE CASES

Fortanix DSM provides multiple interfaces to application developers.

1. **PKCS#11** interface for legacy HSM type integration. Supports **C/C++** programs.
2. **JCE** interface using Fortanix DSM JCE provider, for existing JCE based integrations (For example, replacing BC provider-based code). This applies to **Java** programs.
3. **KMIP** protocol-based interface which can be leveraged using commercial KMIP Clients. Fortanix also provides an inhouse certified **Java** KMIP Client.
4. **RESTful** interface. Fortanix DSM supports host of client SDK's which leverage the REST API interface documented at https://www.fortanix.com/api/ . Programming languages supported are **Java, Python, Golang, C#, Php and JavaScript.**

Details on how to download and install above are mentioned here
https://support.fortanix.com/hc/en-us/sections/360002548231-Downloads

This document focuses on REST API based examples in different languages supported.

3.0 FORTANIX DATA SECURITY MANAGER API ERROR CODES

Fortanix DSM API attempts to return appropriate HTTP status codes for every request.

STATUS CODE	STATUS TEXT	GENERAL DESCRIPTION
200	OK	Success
204	No content	There was no new data to return for this API. However, the execution is successful.
400	Bad request	The request body was invalid for the API. A detailed error description will explain further.
401	Unauthorized	Missing or incorrect authentication credentials.
403	Forbidden	The request is understood, but it has been refused or access is not allowed. A detailed error description will explain further.

STATUS CODE	STATUS TEXT	GENERAL DESCRIPTION
404	Not found	The URI requested is invalid or the resource requested, such as a user, does not exist
409	Conflict	Indicates a request conflict with current state of the server
429	Too many requests	Returned when a request cannot be served due to the app's rate limit having been exhausted for the resource.
500	Internal server error	This is usually a temporary error, for example in a high load situation or if an endpoint is temporarily having issues. Try again and if this is not resolved, raise an issue with Fortanix DSM support team

4.0 CONFIGURE API CLIENT AND CLIENT AUTHENTICATION

The first step is to configure the API client to communicate with the Fortanix DSM service and authenticate with the service.

4.1 CONFIGURE FORTANIX DSM SERVICE ENDPOINT

Fortanix DSM service endpoint need to be configured as the first step. The endpoint is the full Fortanix DSM cluster URL. For example: https://sdkms.fortanix.com

- **Java**

```
ApiClient apiClient = new ApiClient();
apiClient.setBasePath(<Endpoint URL>);
```

- **Python**

```
config = sdkms.v1.Configuration()
config.host = "<Endpoint URL>"
client = sdkms.v1.ApiClient(configuration=config)
```

- **PHP**

```
$configuration = new SdkmsClient\Configuration();
```

```
$configuration->setHost('<Endpoint URL>');
$client = new SdkmsClient\ApiClient($configuration);
```

- **Javascript**

```
var FortanixSdkmsRestApi = require('fortanix_sdkms_rest_api');
var client = FortanixSdkmsRestApi.ApiClient.instance;
client.basePath = '<Endpoint URL>';
```

- **Golang**

```
import (
    "github.com/fortanix/sdkms-client-go/sdkms"
)
client := sdkms.Client{
      Endpoint:   "<Endpoint URL>",
     HTTPClient: http.DefaultClient,
}
```

- **C#**

```
using Fortanix.SDKMS.Client;

Configuration.Default.BasePath = "<Endpoint URL>";
```

- **REST API using curl**

```
curl <Endpoint URL>  ...
```

4.2 CONFIGURE AUTHENTICATION

The method used to configure authentication varies depending on what type of authentication you wish to perform. More details can be found here https://support.fortanix.com/hc/en-us/articles/360015941132-Authentication.

Every authentication creates a session that grants a bearer token which can be passed for any other API requests. Please note that the bearer token expires if the session remains inactive for a long period of time. Please check with Fortanix DSM admin for the expiration values. By default, user sessions expire after 24 hours and app sessions expire after 10 min if remained idle.

The sample code below authenticates using provided credentials and then saves the bearer token in the API Client for subsequent use by other API requests.

4.2.1 USER AUTHENTICATION

Using a user's email and password.

NOTE: If a user is part of multiple accounts, you must first select a particular account, before proceeding to key management operations.

How to get account Id: Go to **Account Settings**. Copy the **Account ID** being displayed on the right.

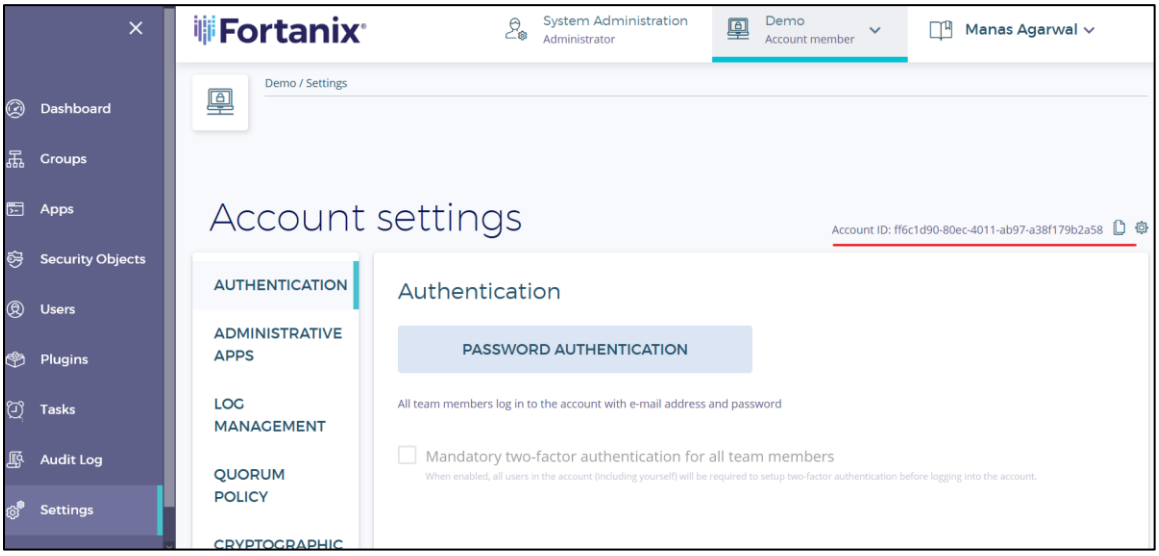

FIGURE 1: GET ACCOUNT ID

- **Java**

```
ApiClient apiClient = new ApiClient();
apiClient.setBasePath(<Endpoint URL>);
apiClient.setUsername(<user email>);
apiClient.setPassword(<user password>);
AuthenticationApi authenticationApi = new AuthenticationApi(apiClient);
AuthResponse authResponse = authenticationApi.authorize();
ApiKeyAuth bearerTokenAuth =
        (ApiKeyAuth) apiClient.getAuthentication("bearerToken");
bearerTokenAuth.setApiKey(authResponse.getAccessToken());
bearerTokenAuth.setApiKeyPrefix("Bearer");

// select account
SelectAccountRequest request = new SelectAccountRequest()
                      .acctId(<Account Id>);
authenticationApi.selectAccount(request);
```

- **Python**

```
config = sdkms.v1.Configuration()
config.host = "<Endpoint URL>"
config.username = <user email>
config.password = <user password>
client = sdkms.v1.ApiClient(configuration=config)
auth_instance = sdkms.v1.AuthenticationApi(api_client=client)
auth = auth_instance.authorize()
config.api_key['Authorization'] = auth.access_token
config.api_key_prefix['Authorization'] = 'Bearer'

// select account
request = sdkms.v1.SelectAccountRequest(<account id>)
auth_instance.select_account(request)
```

- **Golang**

```
client := sdkms.Client{
      Endpoint:   "<Endpoint URL>",
      HTTPClient: http.DefaultClient,
}
ctx := context.Background()
_, err := client. AuthenticateWithUserPass(ctx, <user email>, <user password>)

// select account
request := sdkms.SelectAccountRequest {
            AcctId: <Account Id>
}
_, err = client.SelectAccount(ctx, request)
```

- **C#**

```
Configuration.Default.BasePath = "<Endpoint URL>";
Configuration.Default.Username = <user email>;
Configuration.Default.Password = <user password>;

AuthenticationApi authenticationApi = new AuthenticationApi();
AuthResponse response = authenticationApi.Authorize();
Configuration.Default.AddApiKey("Authorization",
                                response.AccessToken);
Configuration.Default.AddApiKeyPrefix("Authorization", "Bearer");

// select account
SelectAccountRequest request = new SelectAccountRequest()
                        .acctId(<Account Id>);
authenticationApi.SelectAccount(request);
```

- **REST API using curl**

```
$ curl <Endpoint URL>/sys/v1/session/auth -X POST -u <user email>:<user password>
{"token_type":"Bearer","expires_in":600,"access_token":"YhXwwa-6C...L9kRxswmPZkEFQ2ig5g","entity_id":"7916b324-33a1-4a06-8778-59ec0492bb10"}
// select account
$ curl <Endpoint URL>/sys/v1/session/select_account -H 'Authorization: Bearer YhXwwa- 6C...L9kRxswmPZkEFQ2ig5g ' -d '{"acct_id": "<account id>"}'
// use the "access_token" as Bearer Auth in other API requests. E.g:
$ curl <Endpoint URL>/other_apis -H 'Authorization: Bearer YhXwwa-6C...L9kRxswmPZkEFQ2ig5g ' ...
```

4.2.2 APP AUTHENTICATION WITH AN API KEY

- **Java**

```
ApiClient apiClient = new ApiClient();
apiClient.setBasePath(<Endpoint URL>);
apiClient.setBasicAuthString(<API Key>);
AuthenticationApi authenticationApi = new AuthenticationApi(apiClient);
AuthResponse authResponse = authenticationApi.authorize();
ApiKeyAuth bearerTokenAuth =
        (ApiKeyAuth) apiClient.getAuthentication("bearerToken");
bearerTokenAuth.setApiKey(authResponse.getAccessToken());
bearerTokenAuth.setApiKeyPrefix("Bearer");
```

- **Python**

```
config = sdkms.v1.Configuration()
config.host = "<Endpoint URL>"
config.app_api_key = <API Key>
client = sdkms.v1.ApiClient(configuration=config)
auth_instance = sdkms.v1.AuthenticationApi(api_client=client)
auth = auth_instance.authorize()
```

```
config.api_key['Authorization'] = auth.access_token
config.api_key_prefix['Authorization'] = 'Bearer'
```

- **PHP**

```
$configuration = new SdkmsClient\Configuration();
$decodedApiKey = base64_decode('<API Key>')
$decodedApiTokens = explode(":",$decodedApiKey);
$configuration->setUsername($decodedApiTokens[0]);
$configuration->setPassword($decodedApiTokens[1]);
$configuration->setHost('<Endpoint URL>');
$client = new SdkmsClient\ApiClient($configuration);

$api_instance = new SdkmsClient\Api\AuthenticationApi($client);
$auth_response = api_instance.authorize();
$configuration->setApiKey('Authorization', $auth_response-
>getAccessToken());
$configuration->setApiKeyPrefix('Authorization', 'Bearer');
    $auth_response = api_instance.authorize();
```

- **Javascript**

```
var FortanixSdkmsRestApi = require('fortanix_sdkms_rest_api');
var client = FortanixSdkmsRestApi.ApiClient.instance;
client.basePath = '<Endpoint URL>';
let buff = new Buffer('<API Key>;', 'base64');
let decodedApiKey = buff.toString('ascii');
let decodedApiTokens = decodedApiKeytr.split(":");
client.authentications.basicAuth.username = decodedApiTokens[0];
client.authentications.basicAuth.password = decodedApiTokens[1];

var authCallback = function(error, data, response) {
    if (error) {
        console.error("Error: " + JSON.stringify(response));
    } else {
```

```
        client.authentications.bearerToken.apiKey = data.accessToken;
        client.authentications.bearerToken.apiKeyPrefix = "Bearer";
        // continue with other API calls.
    }
};
var authApi = new FortanixSdkmsRestApi.AuthenticationApi()
authApi.authorize(authCallback);
```

- **Golang**

```
client := sdkms.Client{
      Endpoint:   "<Endpoint URL>",
      HTTPClient: http.DefaultClient,
}
ctx := context.Background()
_, err := client. AuthenticateWithAPIKey(ctx, <API Key>)
```

- **C#**

```
Configuration.Default.BasePath = "<Endpoint URL>";
string decodedAPIKey = Encoding.ASCII.GetString(
              Convert.FromBase64String(<API Key>));
string[] decodeApiTokens = decodedAPIKey.Split(':');
Configuration.Default.Username = decodeApiTokens[0];
Configuration.Default.Password = decodeApiTokens[1];

AuthenticationApi authenticationApi = new AuthenticationApi();
AuthResponse response = authenticationApi.Authorize();
Configuration.Default.AddApiKey("Authorization",
                                response.AccessToken);
Configuration.Default.AddApiKeyPrefix("Authorization", "Bearer");
```

- **REST API using curl**

```
$ curl <Endpoint URL>/sys/v1/session/auth -X POST -H 'Authorization:
Basic <API Key>'
 > {"token_type":"Bearer","expires_in":600,"access_token":"YhXwwa-
6C...L9kRxswmPZkEFQ2ig5g","entity_id":"7916b324-33a1-4a06-8778-
59ec0492bb10"}

 // use the "access_token" as Bearer Auth in other API requests. E.g:
$ curl <Endpoint URL>/other_apis -H 'Authorization: Bearer YhXwwa-
6C...L9kRxswmPZkEFQ2ig5g ' ...
```

4.2.3 APP AUTHENTICATION WITH CLIENT CERTIFICATE

This method requires client key and certificate files, along with the app UUID.

- **Java**

The client certificate and client private key must be supplied in a PKCS#12 keystore.

```
// Create PKCS#12 keystore
$ openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem -name
"my-sdkms-app" -out client-sdkms.p12

// Set the keystore in Java program
System.setProperty("javax.net.ssl.keyStoreType", "pkcs12");
System.setProperty("javax.net.ssl.keyStore", </path/to/client-
sdkms.p12>);
System.setProperty("javax.net.ssl.keyStorePassword”, <password of the
PKCS#12 archive>);

ApiClient apiClient = new ApiClient();
apiClient.setBasePath(<Endpoint URL>);
apiClient.setUsername(<App UUID>);
AuthenticationApi authenticationApi = new AuthenticationApi(apiClient);
AuthResponse authResponse = authenticationApi.authorize();
ApiKeyAuth bearerTokenAuth =
       ApiKeyAuth) apiClient.getAuthentication("bearerToken");
```

```
bearerTokenAuth.setApiKey(authResponse.getAccessToken());
bearerTokenAuth.setApiKeyPrefix("Bearer");
```

- **Python**

```
config = sdkms.v1.Configuration()
config.host = "<Endpoint URL>"
config.username = <App UUID>
config.cert_file = <client-cert.pem>
config.key_file = <client-key.pem>
client = sdkms.v1.ApiClient(configuration=config)
auth_instance = sdkms.v1.AuthenticationApi(api_client=client)
auth = auth_instance.authorize()
config.api_key['Authorization'] = auth.access_token
config.api_key_prefix['Authorization'] = 'Bearer'
```

- **Golang**

```
certFile := "client-crt.pem"
keyFile := "client-key.pem"
cert, err := tls.LoadX509KeyPair(*certFile, *keyFile)
caCertPool := x509.NewCertPool()
tlsConfig := &tls.Config{
        Certificates: []tls.Certificate{cert},
        RootCAs:      caCertPool,
}
transport := &http.Transport{TLSClientConfig: tlsConfig}
http_client := &http.Client{Transport: transport}
ctx := context.Background()
client := sdkms.Client{
      Endpoint:   "<Endpoint URL>",
      HTTPClient: &http_client,
}
_, err := client. AuthenticateWithUserPass(ctx, <App UUID>, "")
```

- **C#**

```
 // Create PKCS#12 keystore
$ openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem -name
"my-sdkms-app" -out client-sdkms.p12
// Above asks for password to be set. Note this password

// C# code
using System.Security.Cryptography.X509Certificates;
X509Certificate2 certificate = new X509Certificate2("client-sdkms.p12",
<pkcs12-keystore-pass>);
Configuration.Default.BasePath = "<Endpoint URL>";
Configuration.Default.Username = "<App UUID>";
Configuration.Default.ApiClient.RestClient.ClientCertificates =
         new X509CertificateCollection() { certificate };

AuthenticationApi authenticationApi = new AuthenticationApi();
AuthResponse response = authenticationApi.Authorize();
Configuration.Default.AddApiKey("Authorization",
                                  response.AccessToken);
Configuration.Default.AddApiKeyPrefix("Authorization", "Bearer");
```

- **REST API using curl**

```
$ curl <Endpoint URL>/sys/v1/session/auth -X POST -u <App UUID>: --cert
<client-cert.pem> --key <client-key.pem>
  > {"token_type":"Bearer","expires_in":600,"access_token":"YhXwwa-
6C...L9kRxswmPZkEFQ2ig5g","entity_id":"7916b324-33a1-4a06-8778-
59ec0492bb10"}
// if prompts for password, just press enter


// use the "access_token" as Bearer Auth in other API requests. E.g:
$ curl <Endpoint URL>/other_apis -H 'Authorization: Bearer YhXwwa-
6C...L9kRxswmPZkEFQ2ig5g ' --cert <client-cert.pem> --key <client-
key.pem> ...
```


The user's password, API key, or client certificate private key will grant access to all the keys and methods that the user or application has access to. You should protect these like other sensitive information and not store these in the scripts.

4.3 LOG OUT

Bearer tokens are valid for a limited period of time. User bearer tokens will expire after 24 hours of inactivity, and application bearer tokens will expire after 10 minutes of inactivity. An application may terminate its session to immediately invalidate its bearer token.


It is a good idea to do this before an application exits to limit the possibility of a stolen bearer token being used to access the Fortanix DSM server.

- **Java**

```
AuthenticationApi authenticationApi = new AuthenticationApi(apiClient);
authenticationApi.terminate();
```

- **Python**

```
api_instance = sdkms.v1.AuthenticationApi(api_client=client)
api_instance.terminate()
```

- **PHP**

```
public function logout() {
    $authenticationApi = new Swagger\Client\Api\AuthenticationApi($client);
    $authInstance->terminate();
}
```

- **Javascript**

```
var logoutCallback = function(error, data, response) {
    if (error) {
        console.error("Error: " + JSON.stringify(response));
    } else {
```

```
        console.log('logout successfully');
    }
};

var authenticationApi = new FortanixSdkmsRestApi.AuthenticationApi()
authenticationApi.terminate(logoutCallback);
```

- **Golang**

```
client := sdkms.Client{
          Endpoint:   "<Endpoint URL>",
          HTTPClient: http.DefaultClient,
}
ctx := context.Background()
_, err := client. AuthenticateWithAPIKey(ctx, <API Key>)
client.TerminateSession(ctx)
```

- **C#**

```
AuthenticationApi authenticationApi = new AuthenticationApi();
AuthResponse response = authenticationApi.Terminate();
```

- **REST API using curl**

```
curl <Endpoint URL>/sys/v1/session/terminate -H 'Authorization:
Bearer YhXwwa- 6C...L9kRxswmPZkEFQ2ig5g'
```

5.0 SECURITY OBJECTS DETAILS

5.1 LISTING ALL SECURITY OBJECTS

The `getSecurityObjects` API returns a list of all security objects that the currently authorized entity can access. This supports some filter and sort parameters.

- **Java**

```
SecurityObjectsApi sobjectsApi = new SecurityObjectsApi();
List<KeyObject> allKeys = sobjectsApi.getSecurityObjects(null, null, null);
```

- **Python**

```
api_instance = sdkms.v1.SecurityObjectsApi(api_client=client)
sobjects = api_instance.get_security_objects()
```

- **Golang**

```
keys, err := client.ListSobjects(ctx, nil)
```

- **C#**

```
SecurityObjectsApi sObjectApi = new SecurityObjectsApi();
List<KeyObject> sObjects = sObjectApi.GetSecurityObjects();
```

- **REST API using curl**

```
curl <Endpoint URL>/crypto/v1/keys -H 'Authorization: Bearer YhXwwa-6C...ig5g'
```

5.2 GET A SECURITY OBJECT BY NAME

The `getSecurityObjects` API can also be used to lookup keys by name. The first parameter of this method restricts the returned value to only the security object with the requested name.

- **Java**

```
List<KeyObject> sobjects = sobjectsApi.getSecurityObjects("example-key",
null, null, null);
KeyObject matched = sobjects.get(0); // Max 1 object will be there
                                          in the result list
```

- **Python**

```
api_instance = sdkms.v1.SecurityObjectsApi(api_client=client)
sobjects = api_instance.get_security_objects(name = "example-key")
matched = sobjects[0]
```

- **Golang**

```
keyName := "example-key"
sobjectDescriptor := sdkms.SobjectDescriptor {
    Name: &keyName,
}
key, err := client.GetSobject(ctx, nil, &sobjectDescriptor)
```

- **C#**

```
SecurityObjectsApi sObjectApi = new SecurityObjectsApi();
List<KeyObject> sObjects = sObjectApi.GetSecurityObjects(name : "example-key");
KeyObject matched = sObjects[0];
```

- **REST API using curl**

```
curl <Endpoint URL>/crypto/v1/keys?name=example-key -H 'Authorization: Bearer YhXwwa-6C...ig5g'
```

5.3 GET A SECURITY OBJECT BY UUID

The `getSecurityObject` API fetches security object details by the Object UUID (Key Id).

- **Java**

```
KeyObject sobject = sobjectsApi.getSecurityObject(<Key-UUID>);
```

- **Python**

```
api_instance = sdkms.v1.SecurityObjectsApi(api_client=client)
sobject = api_instance.get_security_object(<Key-UUID>)
```

- **Golang**

```
keyId := "<Key-UUID>"
sobjectDescriptor := sdkms.SobjectDescriptor {
    Kid: &keyId,
}
key, err := client.GetSobject(ctx, nil, &sobjectDescriptor)
```

- **C#**

```
SecurityObjectsApi sObjectApi = new SecurityObjectsApi();
KeyObject key = sObjectApi.GetSecurityObject("<Key-UUID>");
```

- **REST API using curl**

```
curl <Endpoint URL>/crypto/v1/key/<Key-UUID> -H 'Authorization: Bearer
YhXwwa-6C...ig5g'
```

5.4 ERROR CODES

STATUS CODE	STATUS TEXT	GENERAL DESCRIPTION
404	Invalid API path: invalid UUID: Invalid length; expecting 32 or 36 chars, found 9	The UUID passed is not a valid ID.
404	Sobject does not exist	Requested Security Object does not exist as per the authenticated entity

6.0 GENERATING SECURITY OBJECTS

Fortanix DSM can generate new security objects (keys and MAC secrets) with random bytes.

Creating keys is performed with a `generateSecurityObject` API. The API input body, `SobjectRequest` objects define the properties of the key that will be created. The `ObjectType`, `KeySize`, and `EllipticCurve` properties of the `SobjectRequest` determine what type of key will be created. The `name` property is required and must be unique.

If you do not override the default enabled operations, the new key will be supporting all operations that make sense for the type of key. By default, security objects will not have the "Export" operation enabled. So, for example, RSA keys will have the Sign, Verify, Encrypt, Decrypt,

WrapKey, UnwrapKey, DeriveKey and AppManageable operations. They will not have the MacGenerate or MacVerify operations since those operations are not defined for RSA keys.

 **WARNING**: Enabled operations may be removed from keys, but they cannot be added.

The enabled operations are specified using the `KeyOps` property of the `SobjectRequest`. This property is a List. The generated key will be created with its enabled operations equal to the list provided.

If you want to create keys that can be exported from Fortanix DSM, you will need to request that the key be created with the `Export` operation enabled along with any other operations you wish to enable on the key.

For example, to create a 2048 RSA key that is exportable and may only be used for signing and verifying signature, use this `SobjectRequest` as shown below:

- **Java**

```
// Generate RSA Key
SobjectRequest sobjectRequest = new SobjectRequest()
          .name("Name")
          .keySize(2048)
          .objType(ObjectType.RSA);
          .keyOps(Arrays.asList(KeyOperations.SIGN,
                                KeyOperations.VERIFY,
                                KeyOperations.EXPORT));
SecurityObjectsApi securityObjectsApi = new
                           SecurityObjectsApi(apiClient);
KeyObject keyObject =
        securityObjectsApi.generateSecurityObject(sobjectRequest);


// Generate AES Key
SobjectRequest sobjectRequest = new SobjectRequest()
          .name("Name")
          .keySize(128)
```

```
          .objType(ObjectType.AES);
SecurityObjectsApi securityObjectsApi = new
                            SecurityObjectsApi(apiClient);
KeyObject keyObject =
        securityObjectsApi.generateSecurityObject(sobjectRequest);

  // Generate EC Key
SobjectRequest sobjectRequest = new SobjectRequest()
           .name("Name")
           .ellipticCurve(EllipticCurve.NISTP256)
          .objType(ObjectType.EC);
SecurityObjectsApi securityObjectsApi = new
                            SecurityObjectsApi(apiClient);
KeyObject keyObject =
        securityObjectsApi.generateSecurityObject(sobjectRequest);
```

- **Python**

```
api_instance = sdkms.v1.SecurityObjectsApi(api_client=client)

// Generate RSA Key
request = sdkms.v1.SobjectRequest(name='Name', key_size=2048,
                        obj_type= sdkms.v1.ObjectType.RSA)
key = api_instance.generate_security_object(request)

// Generate AES Key
request = sdkms.v1.SobjectRequest(name='Name', key_size=128,
                        obj_type= sdkms.v1.ObjectType.AES)
key = api_instance.generate_security_object(request)

// Generate EC Key
request = sdkms.v1.SobjectRequest(name='Name',
            elliptic_curve= sdkms.v1.EllipticCurve.NISTP256,
            obj_type= sdkms.v1.ObjectType.EC)
key = api_instance.generate_security_object(request)
```

- **PHP**

```
public function generateKey() {
    $securityObjectApi = new Swagger\Client\Api\SecurityObjectsApi($client);
    $ObjectobjType = new Swagger\Client\Model\ObjectType();
    $securityObjectRequest = array('name' => 'Name', 'key_size' => 128,
 'obj_type' => $objType::AES);
    $securityObjectResponse = $securityObjectApi-
 >generateSecurityObject($request);
}
```

- **Javascript**

```
var generateKeyCallback = function(error, data, response) {
    if (error) {
        console.error("Error: " + JSON.stringify(response));
    } else {
        console.log('Security Object Create: ' + JSON.stringify(data));
    }
};

var securityObjectApi = new FortanixSdkmsRestApi.SecurityObjectsApi()
var securityObjectRequest =
FortanixSdkmsRestApi.SobjectRequest.constructFromObject({"name": "Name",
"key_size": 128, "obj_type": "AES"})
securityObjectApi.generateSecurityObject(securityObjectRequest,
generateKeyCallback);
```

- **Golang**

```
// Generate RSA Key
keyName := "name"
objType := sdkms.ObjectTypeRsa
keySize := 2048
sobjectReq := sdkms.SobjectRequest{
      Name:    &name,
```

```
      ObjType: &objType,
      KeySize: &keySize,
}
sobject, err := client.CreateSobject(ctx, sobjectReq)

// Generate AES Key
objType := sdkms.ObjectTypeAes
keySize := 256
sobjectReq := sdkms.SobjectRequest{
      Name:    &name,
      ObjType: &objType,
      KeySize: &keySize,
}
sobject, err := client.CreateSobject(ctx, sobjectReq)

// Generate EC Key
keyName := "name"
objType := sdkms.ObjectTypeEc
curve := sdkms.EllipticCurveNistP256
sobjectReq := sdkms.SobjectRequest{
      Name:    &name,
      ObjType: &objType,
      EllipticCurve: &curve,
}
sobject, err := client.CreateSobject(ctx, sobjectReq)
```

- **C#**

```
SecurityObjectsApi sObjectApi = new SecurityObjectsApi();
// Generate RSA Key
SobjectRequest sobjectRequest = new SobjectRequest(
       Name: "Name",
       KeySize: 2048,
       ObjType: ObjectType.RSA,
       KeyOps: new List<KeyOperations>() { KeyOperations.SIGN,
```

```
                                    KeyOperations.VERIFY,
                                    KeyOperations.EXPORT});
KeyObject keyObject =
            sObjectApi.GenerateSecurityObject(sobjectRequest);

// Generate AES Key
SobjectRequest sobjectRequest = new SobjectRequest(
       Name: "Name",
       KeySize: 128,
       ObjType: ObjectType.AES);

KeyObject keyObject =
          sObjectApi.GenerateSecurityObject(sobjectRequest);

// Generate EC Key
SobjectRequest sobjectRequest = new SobjectRequest()
       Name: "Name",
       EllipticCurve: EllipticCurve.NISTP256,
       ObjType: ObjectType.EC);
KeyObject keyObject =
        sObjectApi.GenerateSecurityObject(sobjectRequest);
```

- **REST API using curl**

```
// Generate RSA Key
curl <Endpoint URL>/crypto/v1/key -H 'Authorization: Bearer YhXwwa-
6C...ig5g' -d '{"name": "Name", "key_size": 2048, "obj_type": "RSA"}'

// Generate AES Key
curl <Endpoint URL>/crypto/v1/key -H 'Authorization: Bearer YhXwwa-
6C...ig5g' -d '{"name": "Name", "key_size": 128, "obj_type": "AES"}'

// Generate EC Key
```

```
curl <Endpoint URL>/crypto/v1/key -H 'Authorization: Bearer YhXwwa-
6C...ig5g' -d '{"name": "Name", "elliptic_curve": "NISTP256",
"obj_type": "EC"}'
```

6.1 ERROR CODES

STATUS CODE	STATUS TEXT	GENERAL DESCRIPTION
400	Miscellaneous error text	The provided input for key generation API is not as expected. The error message will explain the issue.

7.0 IMPORTING SECURITY OBJECTS

Fortanix DSM can import keys, certificates and secrets that were created outside of Fortanix DSM. Note that except for public keys and certificates, it is generally more secure to create keys inside of Fortanix DSM as described in the previous section *"Generating Security Objects"*.

The key value must be unencrypted (that is, not encrypted or password-protected). If the key you are importing is an encrypted key, you must first decrypt the key, either in your application or using a program such as `openssl`. Alternatively, you may import wrapped keys into Fortanix DSM as described in the section *"Unwrapping Security Objects"*.

Importing keys is performed with an `importSecurityObject` API. The `SobjectRequest` object defines the properties of the key that will be imported, including the key material to be used for the key. The `ObjectType` property of the `SobjectRequest` determine what type of key will be imported. The `KeySize` and `ellipticCurve` properties of the `SobjectRequest` are ignored, as the key size or curve can be determined from the key material. The `name` property is required and must be unique. The `value` property provides the key material. The required format of value depends on what type of object is being imported.

If you do not override the default enabled operations, the imported key will be supporting all operations that make sense for the type of key. By default, imported keys will have the Export operation enabled too. So, for example, RSA keys will have the Sign, Verify, Encrypt, Decrypt,

WrapKey, UnwrapKey, DeriveKey and AppManageable operations. They will not have the MacGenerate or MacVerify operations, since those operations are not defined for RSA keys.

WARNING: Enabled operations may be removed from keys, but they cannot be added.

The enabled operations are specified using the `keyOps` property of the `SobjectRequest`. This property is a List. The imported key will be created with its enabled operations equal to the list provided.

If you want to import keys that can be exported from Fortanix DSM, you will need to request that the key be imported with the Export operation enabled along with any other operations you wish to enable on the key.

For example, to create a 2048 RSA key that is exportable and may only be used for signing and verifying the signature, use this `SobjectRequest`:

- **Java**

```
// Import RSA Key
SobjectRequest sobjectRequest = new SobjectRequest()
          .name("Name").value(<key value as bytes[]>)
          .objType(ObjectType.RSA);
          .keyOps(Arrays.asList(KeyOperations.SIGN,
                                KeyOperations.VERIFY,
                                KeyOperations.EXPORT));
SecurityObjectsApi securityObjectsApi = new
                                SecurityObjectsApi(apiClient);
KeyObject keyObject =
        securityObjectsApi.importSecurityObject(sobjectRequest);
// Import Certificate
SobjectRequest sobjectRequest = new SobjectRequest()
          .name("Name").value(<certificate value as bytes[]>)
          .objType(ObjectType.CERTIFICATE);
SecurityObjectsApi securityObjectsApi = new
```

```
                            SecurityObjectsApi(apiClient);
KeyObject keyObject =
        securityObjectsApi.importSecurityObject (sobjectRequest);

// Import Secret
SobjectRequest sobjectRequest = new SobjectRequest()
          .name("Name").value(<secret value as bytes[]>)
          .objType(ObjectType.SECRET);
SecurityObjectsApi securityObjectsApi = new
                            SecurityObjectsApi(apiClient);
KeyObject keyObject =
        securityObjectsApi.importSecurityObject (sobjectRequest);
```

- **Python**

```
api_instance = sdkms.v1.SecurityObjectsApi(api_client=client)

// Import RSA Key
request = sdkms.v1.SobjectRequest(name='Name', value=<key value as
bytes>, obj_type= sdkms.v1.ObjectType.RSA)
key = api_instance.import_security_object(request)
```

- **Golang**

```
// Import RSA Key
objType := sdkms.ObjectTypeRsa
value := []byte(<key value as bytes>)
sobjectReq := sdkms.SobjectRequest{
      Name:    &name,
      ObjType: &objType,
      Value: &value,
}
sobject, err := client.ImportSobject(ctx, sobjectReq)
```

- **C#**

```
SecurityObjectsApi securityObjectsApi = new
                              SecurityObjectsApi(apiClient);
// Import RSA Key
SobjectRequest sobjectRequest = new SobjectRequest(
          Name: "Name"),
          Value: (<key value as bytes[]>),
          ObjType: ObjectType.RSA);
KeyObject keyObject =
        securityObjectsApi.ImportSecurityObject(sobjectRequest);

// Import Secret
SobjectRequest sobjectRequest = new SobjectRequest(
        Name: "Name"),
        Value: (<secret value as bytes[]>),
        ObjType: ObjectType.SECRET);
KeyObject keyObject =
      securityObjectsApi.ImportSecurityObject(sobjectRequest);
```

- **REST API using curl**

```
#Generate RSA Key
curl <Endpoint URL>/crypto/v1/key -H 'Authorization: Bearer YhXwwa-
6C...ig5g' -X PUT -d '{"name": "Name", "value ": <base64-encoded-key-
value>, "obj_type": "RSA"}'
```

7.1 ERROR CODES

STATUS CODE	STATUS TEXT	GENERAL DESCRIPTION
400	Miscellaneous error text	The provided input for key generation API is not as expected. The error message will explain the issue.

8.0 EXPORTING SECURITY OBJECT

Security Object need to be marked as **exportable** for this operation.

- **Java**

```
SobjectDescriptor soDescriptor = new SobjectDescriptor()
            .name(<name of the key>);
KeyObject keyObject = securityObjectsApi.getSecurityObjectValue (soDescriptor);
keyObject.value // contains the key material
```

- **Python**

```
api_instance = sdkms.v1.SecurityObjectsApi(api_client=client)
request = sdkms.v1.SobjectDescriptor(name='<name of the key>')
key = api_instance.get_security_object_value(request)
key.value #contains the key material
```

- **PHP**

```
$api_instance = new SdkmsClient\Api\SecurityObjectsApi($client);
$request=array('name' => '<name of the key>');
$keyObject = api_instance->getSecurityObjectValueEx($request);
$keyObject.value // contains the key material
```

- **Javascript**

```
Var request =
FortanixSdkmsRestApi.SobjectDescriptor.constructFromObject({"name": "<name of
the key>"});
var exportCallback = function(error, data, response) {
    if (error) {
        console.error("Error: " + JSON.stringify(response));
    } else {
        data.value // contains the key material
    }
};
var sObjectsApi = new FortanixSdkmsRestApi.SecurityObjectsApi()
sObjectsApi.getSecurityObjectValueEx(request, exportCallback);
```

- **Golang**

```
keyName := "<name of the key>"
sobject, err := client.ExportSobject(ctx,
                        sdkms.SobjectByName(keyName))
```

- **C#**

```
SobjectDescriptor soDescriptor = new SobjectDescriptor(
            Name: (<name of the key>);
KeyObject keyObject = securityObjectsApi.GetSecurityObjectValue (soDescriptor);
keyObject.Value // contains the key material
```

- **REST API using curl**

```
curl <Endpoint URL>/crypto/v1/keys/export -H 'Authorization: Bearer
YhXwwa-6C...ig5g' -d '{"name": "Key-Name"}'
```

8.1 ERROR CODES

STATUS CODE	STATUS TEXT	GENERAL DESCRIPTION
403	Requested operation is not allowed with this key	This key cannot be exported. This may be due to unauthorized access, key not being marked as exportable, or the key being in destroyed or deleted state.

9.0 DELETING SECURITY OBJECT

- **Java**

```
KeyObject keyObject = securityObjectsApi.deleteSecurityObject(<Key-UUID>);
```

- **Python**

```
api_instance = sdkms.v1.SecurityObjectsApi(api_client=client)
api_instance.delete_security_object_value(<Key-UUID>)
```

- **Golang**

```
keyId := "<Key-UUID>"
err := client. DeleteSobject(ctx, *keyId)
```

- **C#**

```
KeyObject keyObject = securityObjectsApi.DeleteSecurityObject (<Key-UUID>);
```

- **REST API using curl**

```
curl -X DELETE <Endpoint URL>/crypto/v1/keys/<Key-UUID> -H
'Authorization: Bearer YhXwwa-6C...ig5g'
```

10.0 ENCRYPTION

10.1 AES ENCRYPTION

Encryption using an AES Key requires the following parameters to be set:

- **Alg**: Same as the key type: *AES*
- **Mode**: One of the supported values: ECB, CBC, CBCNOPAD, CFB, CTR, OFB, GCM, CCM, KW, KWP
- **IV**: Initialization Vector can be passed when *Mode* is one of the values: CBC, CBCNOPAD, CFB, CTR, OFB, GCM, CCM. If IV is not passed, then Fortanix DSM will generate a random IV for the operation and return in the response. One should note this IV for decryption purposes later.
- **Tag Len**: For mode GCM and CCM, tag length needs to be passed.
- **AD**: For mode GCM and CCM, Authentication Data needs to be passed.

- **Java**

```
String data = "Hello World!";
byte[] plain = data.getBytes();
EncryptRequest encryptRequest = new EncryptRequest();
encryptRequest
        .alg(ObjectType.AES)
        .plain(plain)
        .mode(CryptMode.CBC);
```

```
EncryptResponse encryptResponse =
    encryptionAndDecryptionApi.encrypt(<Key UUID>, encryptRequest);
encryptResponse.cipher // encrypted data in bytes[]
encryptResponse.iv     // Initialization vector
```

- **Python**

```
api_instance = sdkms.v1.EncryptionAndDecryptionApi(api_client=client)
data = "Hello World!"
request = sdkms.v1.EncryptRequest(
            alg=ObjectType.AES,
            plain= bytearray (data),
            mode=CipherMode.CBC)
encryption_response = api_instance.encrypt(<Key UUID>, request)
encryption_response.cipher # encrypted data in bytearray
encryption_response.iv     # Initialization vector
```

- **PHP**

```
public function encrypt() {
    $cryptMode = new Swagger\Client\Model\CryptMode();
    $encryptionRequestBody = array('alg' => $objType::AES, 'mode' =>
 $cryptMode::CBC, 'plain' => $plain);
    $encryptionRequest = new
 Swagger\Client\Model\EncryptRequest($encryptionRequestBody);
    $encryptionAndDecryptionApi = new
 Swagger\Client\Api\EncryptionAndDecryptionApi($client);
    $encryptionResponse = $encryptionAndDecryptionApi-
 >encrypt($securityObjectResponse["kid"], $encryptionRequest);
}
```

- **Javascript**

```
var encryptCallback = function(error, data, response) {
    if (error) {
```

```
        console.error("Error: " + JSON.stringify(response));
    } else {
        console.log('Data encrypted successfully. result: '
 JSON.stringify(data));
    }
};

var encryptApi = new
FortanixSdkmsRestApi.EncryptionAndDecryptionApi()
var plain = btoa("Hello World!")
var encryptRequest =
FortanixSdkmsRestApi.EncryptRequest.constructFromObject({"alg"
:"AES", "plain": plain, "mode": "CBC"})
encryptApi.encrypt(data["kid"], encryptRequest, encryptCallback)
```

- **Golang**

```
data := byte[]("Hello World!")
keyId := <Key UUID>
encryptReq := sdkms.EncryptRequest{
            Plain: byte[]("Hello World!"),
            Alg:   sdkms.AlgorithmAes,
            Key:   sdkms.SobjectById(keyId),
            Mode:  sdkms.CryptModeSymmetric(sdkms.CipherModeCbc),
}
encryptResp, err := client.Encrypt(ctx, encryptReq)
encryptResp.Cipher // encrypted data as bytes
encryptResp.Iv     // Initialization vector
```

- **REST API using curl**

```
$ echo "Hello World!" | base64
SGVsbG8gV29ybGQhCg==
```

```
$ curl <Endpoint URL>/crypto/v1/encrypt -H 'Authorization: Bearer
YhXwwa-6C...ig5g' -d '{"key": {"kid": "Key-UUID"}, "alg": "AES",
"mode": "CBC", "plain": "SGVsbG8gV29ybGQhCg=="}'

{"cipher": " YiBmaHViIGNpdXJlZyB1eXZpZyB2ZQoK", "iv":
"Y25lYm4gdmVidmllamJ2ZWlqYgo="}
```

10.2 RSA ENCRYPTION

Encryption using an RSA Key requires the following parameters to be set:

- **Alg**: Same as the key type: *RSA*
- **Mode**: One of the supported values: PKCS1_V15, OAEP_MGF1_SHA1, OAEP_MGF1_SHA256, OAEP_MGF1_SHA384, OAEP_MGF1_SHA512

NOTE: RSA key cannot encrypt data that is larger than the size of the key. For example, with RSA 1024 key, one can encrypt data not larger than 1024 bits.

- **Java**

```
String data = "Hello World!";
byte[] plain = data.getBytes();
EncryptRequest encryptRequest = new EncryptRequest();
encryptRequest
        .alg(ObjectType.RSA)
        .plain(plain)
        .mode(CryptMode.OAEP_MGF1_SHA1);
EncryptResponse encryptResponse =
    encryptionAndDecryptionApi.encrypt(<Key UUID>, encryptRequest);
encryptResponse.cipher // encrypted data in bytes[]
```

- **Python**

```
api_instance = sdkms.v1.EncryptionAndDecryptionApi(api_client=client)
data = "Hello World!"
request = sdkms.v1.EncryptRequest(
```

```
            alg=ObjectType.RSA,
            plain= bytearray (data),
            mode=CipherMode.OAEP_MGF1_SHA1)
encryption_response = api_instance.encrypt(<Key UUID>, request)
encryption_response.cipher # encrypted data in bytearray
```

- **Golang**

```
padding := sdkms.RsaEncryptionPaddingOAEPMGF1(
                sdkms.DigestAlgorithmSha1)
encryptReq := sdkms.EncryptRequest{
             Plain: byte[]("Hello World!"),
             Alg:   sdkms.AlgorithmRsa,
             Key:   sdkms.SobjectById(<Key UUID>),
             Mode:  sdkms.CryptModeRSA(padding),
      }
encryptResp, err := client.Encrypt(ctx, encryptReq)
encryptResp.Cipher // encrypted data as bytes
encryptResp.Iv     // Initialization vector
```

- **REST API using curl**

```
$ echo "Hello World!" | base64
SGVsbG8gV29ybGQhCg==

$ curl <Endpoint URL>/crypto/v1/encrypt -H 'Authorization: Bearer
YhXwwa-6C...ig5g' -d '{"key": {"kid": "Key-UUID"}, "alg": "RSA",
"mode": "OAEP_MGF1_SHA1", "plain": "SGVsbG8gV29ybGQhCg=="}'

{"cipher": " YiBmaHViIGNpdXJl…ZyB1eXZpZyB2ZQoK"}
```

11.0 DECRYPTION

Follow similar input parameters used for encryption. See the previous section.

- **Java**

```
byte[] cipher, iv;
DecryptRequest encryptRequest = new DecryptRequest();
decryptRequest
       .alg(ObjectType.AES)
       .cipher(cipher)
       .mode(CryptMode.CBC)
       .iv(iv);
DecryptResponse decryptResponse =
    encryptionAndDecryptionApi.decrypt(<Key UUID>, decryptRequest);
decryptResponse.plain // decrypted plaintext data
```

- **Python**

```
cipher, iv
api_instance = sdkms.v1.EncryptionAndDecryptionApi(api_client=client)
request = sdkms.v1.DecryptRequest(
       alg=ObjectType.AES, cipher= cipher, iv=iv, mode=CipherMode.CBC)
decryption_response = api_instance.decrypt(<Key UUID>, request)
decryption_response.plain # decrypted plain text data
```

- **PHP**

```
Public function decrypt() {
    $decryptionRequestBody = array('alg' => $objType::AES, 'mode' =>
 $cryptMode::CBC, 'cipher' => $encRes['cipher'], 'iv' =>
 $encRes['iv']);
    $decryptionRequest = new
 Swagger\Client\Model\DecryptRequest($decryptionRequestBody);
    $encryptionAndDecryptionApi = new
 Swagger\Client\Api\EncryptionAndDecryptionApi($client);
   $decryptionResponse = $encryptionAndDecryptionApi-
>decrypt($securityObjectResponse["kid"], $decryptionRequest); }
```

- **Javascript**

```
var decryptCallback = function(error, data, response) {
    if (error) {
        console.error("Error: " + JSON.stringify(response));
    } else {
        console.log('Cipher decrypted successfully. result: ' +
 JSON.stringify(data));
    }
};

var encryptionAndDecryptionApi = new
FortanixSdkmsRestApi.EncryptionAndDecryptionApi()
var decryptRequest =
FortanixSdkmsRestApi.DecrypttRequest.constructFromObject({"alg":
"AES", "mode": "CBC", "cipher": "cipher"})
encryptionAndDecryptionApi.decrypt(data["kid"], decryptRequest,
decryptCallback)
```

- **Golang**

```
iv := byte[](<iv bytes>)
keyId := <Key UUID>
decryptReq := sdkms.DecryptRequest{
            Cipher: byte[](<cipher in bytes>),
            Iv:    &iv
            Alg:   sdkms.AlgorithmAes,
            Key:   sdkms.SobjectById(keyId),
            Mode:  sdkms.CryptModeSymmetric(sdkms.CipherModeCbc),
      }
decryptResp, err := client.Decrypt(ctx, decryptReq)
decryptResp.Plain // decrypted plain text data
```

- **REST API using curl**

```
$ curl <Endpoint URL>/crypto/v1/encrypt -H 'Authorization: Bearer
YhXwwa-6C...ig5g' -d '{"key": {"kid": "Key-UUID"}, "alg": "AES",
```

```
"mode": "CBC", "cipher": "YiBmaHViIGNpdXJlZyB1eXZpZyB2ZQoK", "iv":
"Y25lYm4gdmVidmllamJ2ZWlqYgo="}'
{"plain": "SGVsbG8gV29ybGQhCg=="}

$ echo "SGVsbG8gV29ybGQhCg==" | base64 -d
Hello World!
```

12.0 SIGNATURE GENERATION

Supported by RSA and EC type of keys. For signature, one can pass one of the following inputs:

- **Data and Hash Alg**: Pass in the entire data to be signed with the Hash Algorithm to be used for hashing (SHA1, SHA256, SHA512 etc)
- **Hash**: Only pass the already hashed data. Also pass the Hash Algorithm used to generate the said hash.

- **Java**

```
// With data
String data = "Hello World!";
SignRequest signatureRequest = new SignRequest()
          .hashAlg(DigestAlgorithm.SHA256)
          .data(data.getBytes());
SignResponse signResponse =
             SignAndVerifyApi().sign(<Key UUID>, signatureRequest);
signResponse.signature

// With hash
SignRequest signatureRequest = new SignRequest()
          .hashAlg(DigestAlgorithm.SHA256)
          .hash(hash);
SignResponse signResponse =
             SignAndVerifyApi().sign(<Key UUID>, signatureRequest);
signResponse.signature
```

- **Python**

```
api_instance = sdkms.v1.SignAndVerifyApi (api_client=client)
String data = "Hello World!";
request = sdkms.v1.SignRequest(hash_alg= DigestAlgorithm.SHA256,
         data=data.encode())
sign_response = api_instance.sign(<Key UUID>, request)
sign_response.signature

// With hash
request = sdkms.v1.SignRequest(hash_alg= DigestAlgorithm.SHA256, hash=hash)
sign_response = api_instance.sign(<Key UUID>, request)
sign_response.signature
```

- **Golang**

```
data := byte[]("Hello World!")
keyId := <Key UUID>
signReq := sdkms.SignRequest{
            Data:    &data,
            HashAlg: sdkms.DigestAlgorithmSha256,
            Key:     sdkms.SobjectById(keyId),
}
signResp, err := client.Sign(ctx, signReq)
signResp.Signature

// with hash
signReq := sdkms.SignRequest{
            Hash:    &hash,
            HashAlg: sdkms.DigestAlgorithmSha256,
            Key:     sdkms.SobjectById(keyId),
}
signResp.Signature
```

- **REST API using curl**

```
$ echo "Hello World!" | base64
```

```
SGVsbG8gV29ybGQhCg==

$ curl <Endpoint URL>/crypto/v1/sign -H 'Authorization: Bearer
YhXwwa-6C...ig5g' -d '{"key": {"kid": "Key-UUID"}, "hash_alg":
"SHA256", "data": "SGVsbG8gV29ybGQhCg=="}'
{"signature": "Y25lYm4gdmVidm...llamJ2ZWlqYgo=="}
```

13.0 SIGNATURE VERIFICATION

- **Java**

```
String data = "Hello World!";
VerifyRequest verifyRequest = new VerifyRequest()
        .hashAlg(DigestAlgorithm.SHA256)
        .data(data.getBytes())
        .signature(<signature as bytes>;
VerifyResponse verifyResponse =
           SignAndVerifyApi().verify(<Key UUID>, verifyRequest);
verifyResponse.result;   // true or false for verification
```

- **Python**

```
api_instance = sdkms.v1.SignAndVerifyApi (api_client=client)
String data = "Hello World!"
request = sdkms.v1.VerifyRequest(
     hash_alg= DigestAlgorithm.SHA256,
     data=data.encode(),
     signature=<signature in bytes>)
verify_response = api_instance.verify(kid, request)
verify_response.result # true or false for verification
```

- **Golang**

```
data := byte[]("Hello World!")
keyId := <Key UUID>
```

```
verifyReq := sdkms.VerifyRequest{
            Signature: byte[](<signature in bytes>),
            Key:       sdkms.SobjectById(keyId),
            HashAlg:   sdkms.DigestAlgorithmSha256,
            Data:      &data,
}
verifyResp, err := client.Verify(ctx, verifyReq)
verifyResp.Result // true or false for verification
```

- **REST API using curl**

```
$ echo "Hello World!" | base64
SGVsbG8gV29ybGQhCg==

$ curl <Endpoint URL>/crypto/v1/verify -H 'Authorization: Bearer
YhXwwa-6C...ig5g' -d '{"key": {"kid": "Key-UUID"}, "hash_alg":
"SHA256", "data": "SGVsbG8gV29ybGQhCg==", "signature":
"Y25lYm4gdmVidm...llamJ2ZWlqYgo=="}}'

{"result": true}
```

14.0 WRAPPING A KEY

This operation allows a security object to be encrypted by another key for export and transfer out of Fortanix DSM to other systems.

Requirements:

- The target key to be wrapped, need to be marked **Exportable**.
- The wrapping key needs to have **WRAPKEY** operation enabled.
- Symmetric keys (AES, DES, DES3), HMAC keys, Opaque objects, and Secret objects can be wrapped with other symmetric or asymmetric keys.
    - Note: Asymmetric Keys (RSA/DSA), cannot wrap keys/secrets with a size larger than the key size.
- Asymmetric keys (RSA/DSA) can be wrapped with symmetric keys (AES etc) only. Wrapping an asymmetric key with an asymmetric key is **not** supported.

- The wrapping parameters will the follow same guidelines as general Encryption operation by the wrapping key. See the Encryption section for more details.

- **Java**

```
// Wrapping Key with an AES Key
WrapKeyRequest wrapKeyRequest = new WrapKeyRequest()
                  .alg(ObjectType.AES)
                  .kid(<Target Key UUID>)
                  .mode(CryptMode.CBC);
WrappingAndUnwrappingApi wrappingAndUnwrappingApi = new
          WrappingAndUnwrappingApi(apiClient);
WrapKeyResponse wrapKeyResponse = wrappingAndUnwrappingApi
             .wrapKey(<Wrapping Key UUID>, wrapKeyRequest);
wrapKeyResponse.wrappedKey // wrapped key bytes
```

- **Python**

```
// Wrapping Key with an AES Key
api_instance = sdkms.v1.WrappingAndUnwrappingApi(api_client=client)

request = sdkms.v1.WrapKeyRequest(
             alg=ObjectType.AES,
             kid=<target Key UUID>, mode=CryptMode.CBC)
wrapping_response = api_instance
                     .wrap_key(<Wrapping Key UUID, request)
wrapping_response.wrapped_key // wrapped key bytes
```

- **PHP**

```
public function wrapKey() {
    //  kid of key being wrapped
    $wrapKeyRequestBody = array('alg' => $objType::AES, 'mode' =>
 $cryptMode::CBC, 'kid' => kid);
    $wrapKeyRequest = new
 Swagger\Client\Model\WrapKeyRequest($wrapKeyRequestBody);
```

```
    $wrappingAndUnwrappingApi = new
 Swagger\Client\Api\WrappingAndUnwrappingApi($client);
    // kid of wrapping key
    $wrapKeyResponse = $wrappingAndUnwrappingApi->wrapKey(kid,
 $wrapKeyRequest);
}
```

- **Javascript**

```
var wrapKeyCallback = function(error, data, response) {
    if (error) {
        console.error("Error: " + JSON.stringify(response));
    } else {
        console.log('Key wrapped successfully. result: ' +
 JSON.stringify(data));
    }
};

//  kid of key being wrapped
var wrapKeyRequest = new
FortanixSdkmsRestApi.WrapKeyRequest.constructFromObject({"alg":
"AES", "kid": kid, "mode": "CBC"});
var wrappingAndUnwrappingApi = new
FortanixSdkmsRestApi.WrappingAndUnwrappingApi();
// kid of wrapping key
wrappingAndUnwrappingApi.wrapKey(kid, wrapKeyRequest,
wrapKeyCallback);
```

- **Golang**

```
wrapKeyReq := sdkms.WrapKeyRequest {
            Subject: sdkms.SobjectById(<Target Key UUID>),
            Alg:   sdkms.AlgorithmAes,
            Key:   sdkms.SobjectById(<Wrapping Key UUID>),
            Mode:  sdkms.CryptModeSymmetric(sdkms.CipherModeCbc),
```

```
}
wrapKeyResp, err := client.Wrap(ctx, wrapKeyReq)
wrapKeyResp.WrappedKey // wrapped key bytes
```

- **REST API using curl**

```
$ curl <Endpoint URL>/crypto/v1/wrapkey -H 'Authorization: Bearer
YhXwwa-6C...ig5g' -d '{"key": {"kid": "Wrapping-Key-UUID"},
"subject": {"kid": "Target Key UUID"}, "alg": "AES", "mode": "CBC",
"plain": "SGVsbG8gV29ybGQhCg=="}'

{"wrapped_key": "YiBmaHViIGNpdXJl…ZyB1eXZpZyB2ZQoK", "iv" =
"Y25lYm4gdmVidmllamJ2ZWlqYgo="}
```

15.0 UNWRAPPING A KEY

This operation unwraps (decrypts) a wrapped key and import into Fortanix DSM. This allows securely importing into Fortanix DSM security objects that were previously wrapped by Fortanix DSM or another key management system. A new security object will be created in Fortanix DSM with the unwrapped data.

- The `Alg` and `Mode` parameters specify the encryption algorithm and cipher mode being used by the unwrapping key (See Encryption Section).
- The `ObjectType` parameter specifies the object type of the security object being unwrapped. The size or elliptic curve of the object being unwrapped does not need to be specified.

- **Java**

```
// Unwrap an AES key that is wrapped with an RSA key
UnwrapKeyRequest unwrapRequest = new UnwrapKeyRequest()
                .objType(ObjectType.AES)
                .name("new AES key")
                .wrappedKey(<wrapped key in bytes>)
                .alg(ObjectType.RSA); // Unwrapping key type
```

```
KeyObject unwrappedKey = new WrappingAndUnwrappingApi(apiClient)
        .unwrapKey(<UUID of the unwrapping key>, unwrapRequest);
```

- **Python**

```
// Unwrap an AES key that is wrapped with an RSA key
api_instance = sdkms.v1.WrappingAndUnwrappingApi(api_client=client)
request = sdkms.v1.UnwrapKeyRequest(
      alg=ObjectType.RSA,  // Unwrapping Key Type
      obj_type=ObjectType.AES,
      wrapped_key=<wrapped key in bytes>
      name="new AES KEY")
unwrapping_response = api_instance
          .unwrap_key(<UUID of the unwrapping key>, request)
```

- **Golang**

```
newKeyName := "new AES Key"
unwrapKeyReq := sdkms.UnwrapKeyRequest {
            Name: &newKeyName,
            Alg:   sdkms.AlgorithmRsa  // Unwrapping key type
            ObjType:   sdkms.AlgorithmAes,
            WrappedKey:  new byte[](<wrapped key in bytes>),
}
unwrapKeyResp, err := client.Unwrap(ctx, unwrapKeyReq)
```

- **REST API using curl**

```
$ curl <Endpoint URL>/crypto/v1/unwrapkey -H 'Authorization: Bearer
YhXwwa-6C...ig5g' -d '{"key": {"kid": "Unwrapping-Key-UUID"}, "alg":
"RSA", "obj_type": "AES", "wrapped_key": "YiBmal…ZyB1eXZpZyB2ZQoK",
"name": "new AES Key"}'
```

16.0 GROUP MANGEMENT

All group management operations will require User authentication.

For better automation, one can also use an account level administrative app. See this support article for more details https://support.fortanix.com/hc/en-us/articles/360033272171-User-s-Guide-Authentication#1-8-create-applications-and-groups-programmatically-using-app-authentication-methods-0-25

16.1 CREATE GROUP

- **Java**

```
GroupRequest request = new GroupRequest("My Encryption Group");
new GroupsApi(apiClient).createGroup(request);
```

- **Python**

```
api_instance = sdkms.v1.GroupsApi(api_client=client)
request = sdkms.v1.GroupRequest(name="My Encryption Group")
api_instance.create_group(request)
```

- **REST API using curl**

```
$ curl <Endpoint URL>/sys/v1/groups -H 'Authorization: Bearer
YhXwwa-6C...ig5g' -d '{"name": "My Encryption Group"}'
```

16.2 ADD QUORUM APPROVAL POLICY

See this article for more information on quorum approval policies
https://support.fortanix.com/hc/en-us/articles/360016047771-User-s-Guide-Quorum-Policy

Following is an example of adding a simple quorum approval policy to an existing group

- 2 out 3 users for approval (Note the userId of the 3 users)

- **REST API using curl**

```
$ curl <Endpoint URL>/sys/v1/groups/<Group-UUID> -H 'Authorization:
Bearer YhXwwa-6C...ig5g' -X PATCH -d
'{"approval_policy":{"protect_manage_operations":true,"protect_crypt
o_operations":true,"quorum":{"n":2,"members":[{"user":"80e3f312-
9d8e-4645-848a-5c80aebf8f52"},{"user":"ba4ecb63-bcab-4e12-9432-
b0a06fff5226"},{"user":"504c9895-c9fa-4890-b3d5-
35554c74df18"}],"require_2fa":false,"require_password":false}}}'
```

16.3 UPDATE QUORUM APPROVAL POLICY

See this article for more information on quorum approval policies
https://support.fortanix.com/hc/en-us/articles/360016047771-User-s-Guide-Quorum-Policy

For updating an existing quorum approval policy, one need to request for the changes in the policy to be approved.

Following is an example of changing an existing quorum approval policy to add a new user in the policy (2 out 4 users for approval)

- **REST API using curl**

```
$ curl <Endpoint URL>/sys/v1/approval_requests -H 'Authorization:
Bearer YhXwwa-6C...ig5g' \
-d '{"method": "PATCH", "operation": "/sys/v1/groups/<Group-UUID>",
"body":{"approval_policy":{"protect_manage_operations":true,"protect
_crypto_operations":true,"quorum":{"n":2,"members":[{"user":"80e3f31
2-9d8e-4645-848a-5c80aebf8f52"},{"user":"ba4ecb63-bcab-4e12-9432-
b0a06fff5226"},{"user":"504c9895-c9fa-4890-b3d5-35554c74df18"},
{"user":"cbc09134-9a07-4cc2-a59c-845c600e4234"}],
"require_2fa":false,"require_password":false}}}}'

{"request_id": "e8d011db-1a8b-4904-9fcc-1720aa30ac4f" ...}
```

```
// This will create an approval task (with above id). Get it
approved by existing approvers for changes to get reflected.
```

16.4 ADD/UPDATE CRYPTOGRAPHIC POLICY

See this article for more information on cryptographic policies https://support.fortanix.com/hc/en-us/articles/360042064051-User-s-Guide-Cryptographic-Policy

Following is an example of Cryptographic policy which:

- Allows only AES 256 keys.
- Allows only RSA 2048 and above keys.
- No other keys allowed.
- Only supports Encrypt, Decrypt, Sign and Verify operations
- Restricts use of keys not compliant with the policy.

When there is no Quorum Policy on the Group

- **REST API using curl**

```
$ curl <Endpoint URL>/sys/v1/groups/<Group-UUID> -H 'Authorization:
Bearer YhXwwa-6C...ig5g' -X PATCH \
-d '{cryptographic_policy": {"legacy_policy": "prohibited",
"key_ops": ["SIGN", "VERIFY", "ENCRYPT", "DECRYPT"], "aes":
{"key_sizes": [256]}, rsa": { minimum_key_length": 2048}}}'
```

When there is a Quorum Policy enabled on the Group

In this case, you need to use the "approval_request" API.

- **REST API using curl**

```
$ curl <Endpoint URL>/sys/v1/approval_requests -H 'Authorization:
Bearer YhXwwa-6C...ig5g' \
-d ' {"method": "PATCH", "operation": "/sys/v1/groups/<Group-UUID>",
"body":{cryptographic_policy": {"legacy_policy": "prohibited",
```

```
"key_ops": ["SIGN", "VERIFY", "ENCRYPT", "DECRYPT"], "aes":
{"key_sizes": [256]}, rsa": { minimum_key_length": 2048}}}}'
```