Samsung Knox

Samsung Knox Certificate Enrollment Policy API Guide

SAMSUNG

About this guide

This guide describes the Certificate Enrollment Policy APIs flow.

Document Information

This document was last modified on July 4, 2014.

Contents

1. Setup Certificate Enrollment Policy (CEP) Service on Knox device

After the KNOX container creation, the admin can push CEP service (Eg: SCEP, EST-CMC, CMP) APK to device and associate it with container/ownerspace.

For example, if the requirement is to provision and manage certificate for apps inside the container only, then the CEP service must be installed only inside the container. If the requirement is to provision and manage certificate for apps at user space level, then the CEP service can be installed in the user space to provision and manage certificates.

NOTE: The Certificate Enrollment Policy (CEP) service operates only within the scope of container or user space only depending on where it is installed.

MDM agents can call the CEP service present in its own user space or its own container(s) only. MDM agent does not have access to the service created outside its scope (beyond user space or container).

1.1 Pre-requisite

Please note that APK using the below code snippets for various certificate enrollment flows should be an Application signed by Samsung provided Platform Keys.

Let's run through the enrollment using SCEP Protocol (i.e SCEP Service Application).

The APK needs to have the necessary permission in AndroidManifest.xml:
<uses-permission android:name ="
com.sec.enterprise.knox.permission.***KNOX_CERTENROLL***" />
Java file using the code needs following import statements:

```
import com.sec.enterprise.knox.EnterpriseKnoxManager;
import com.sec.enterprise.knox.certenroll.CEPConstants;
import com.sec.enterprise.knox.certenroll.EnterpriseCertEnrollPolicy;
import com.sec.enterprise.knox.certenroll.SCEPProfile;
import com.sec.enterprise.knox.certenroll.EnrollmentProfile;
import com.sec.enterprise.knox.certenroll.TLVUtil;
import com.sec.enterprise.knox.container.KnoxContainerManager;
```


2. Certificate Enrollment

Get an instance of EnterpriseCertEnrollPolicy from the EnterterpriseKnoxManager class. However to get an instance of EnterpriseCertEnrollPolicy for container, KnoxContainerManager class API is being used along with corresponding container ID.

For enrollment of certificate, please refer to code snippet given below:

```
Public void testEnroll(){
Context mContext = this.getApplicationContext();
EnterpriseKnoxManager ekm = null;
KnoxContainerManager kcm = null;
EnterpriseCertEnrollPolicy cep = null;
String mCEPProtocol = CEPConstants.CERT_PROFILE_TYPE_SCEP; // for SCEP Protocol
int mContainerId = 100;           // for container Id 100

//getting CEP policy object inside container
try {
    kcm = EnterpriseKnoxManager.getInstance()
                        .getKnoxContainerManager(mContext, mContainerId);
    cep = kcm.getEnterpriseCertEnrollPolicy(mCEPProtocol);
} catch (NoSuchMethodError e) {
      // TODO Auto-generated catch block
    Log.d(TAG,"Platform doesn't support CEP");
    e.printStackTrace();
}
//getting CEP policy outside the container i.e. inside user space(Owner)
/*
try {
    ekm = EnterpriseKnoxManager.getInstance();
    cep = ekm.getEnterpriseCertEnrollPolicy(mContext, mCEPProtocol);
} catch (NoSuchMethodError e) {
     // TODO Auto-generated catch block
    Log.d(TAG," Platform doesn't support CEP");
    e.printStackTrace();
}
*/

// Setting up SCEP profile
// [scepUrl, subject, alias, KeyStoreType are Mandatory parameters]

EnrollmentProfile enrollmentProfile = new SCEPProfile();
enrollmentProfile.scepUrl = "http://host:port/uri";
enrollmentProfile.scepProfileName = "emailprofile";
enrollmentProfile.setProfileType(CEPConstants.CERT_PROFILE_TYPE_SCEP);
// challenge password from CEP Server(IT-Admin)
enrollmentProfile.challengePassword = "aqadss248723hiuu";
enrollmentProfile.challengeLength = 16; // Challenge byte length
//validity time for the challenge in minutes.
enrollmentProfile.validitytimeForChallenge = 60;
// Subject name for the client certificate.
enrollmentProfile.subjectName = "CN=admin";
// Client Account's email address.
enrollmentProfile.subjectAlterNativeName="user.name@samsung.com";
// Keystore Type is CCM considered in this case.
enrollmentProfile.setKeystoreType(CEPConstants.CEP_KEYSTORETYPE_CCM);
```



```
enrollmentProfile.setKeySize(2048);
enrollmentProfile.setKeyPairAlgorithm (CEPConstants.CEP_KEYALGO_TYPE_RSA);
enrollmentProfile.setCertificateAlias("cert");
boolean enroltype = true; //if CA certificate validation needed

/* “regId” e.g. This is a unique number for each execution and will basically
match to the reference number (refNum) that is available/received in broadcast
receiver explained in NOTE 1. */
String regId = null;

// mList is list of All allowed packages. Please refer NOTE 2 for more details.
List<String> mList = new ArrayList<String>();
mList.add(“com.android.email”); // Email Application is allowed to access “cert”

/* caCertHash is the MD5 hash of the CA Certificate in hex format.
   Refer NOTE 3 for more details.*/
String caCertHash = “13E26F6820EC0689B18A0915F3247D7A”

//Trigger Certificate Enrollment
if(cep != null) {
    if(!enroltype){
        regId=  cep.enrollUserCertificate(enrollmentProfile,
                    mList, null); //null if CA Certhash Validation is not required.
    } else {
        regId=  cep.enrollUserCertificate(enrollmentProfile, mList, caCertHash);
    }
}
}
```

NOTE 1: Agent/Caller need to store the refNum received, along with the Alias used for the enrollment. Corresponding Status, Certificate hash (certHash) and Transaction Id (transactionId) will be received in the broadcast if operation is succeeded as depicted below:

```
public class CertEnrolBindReciever extends BroadcastReceiver {
   public void onReceive(Context context, Intent intent) {
if(intent.getAction().equalsIgnoreCase(CEPConstants.CEP_ACTION_CERT_ENROLL_STATUS))
{
    //used while deletion and renewal i.e testRenew()/testDelete().
    String certHash = intent.getStringExtra(CEPConstants.EXTRA_ENROLL_CERT_HASH);
    //used to query the status of certificate i.e. testgetStatus()
    String transactionId  =
        intent.getStringExtra(CEPConstants.EXTRA_ENROLL_TRANSACTION_ID);
    String alias = intent.getStringExtra(CEPConstants.EXTRA_ENROLL_ALIAS);
    int status = intent.getIntExtra(CEPConstants.EXTRA_ENROLL_STATUS, -1);
    String refNum =
        intent.getStringExtra(CEPConstants.EXTRA_ENROLL_REFERENCE_NUMBER);
    /* Agent/Caller can store the necessary data like Alias, certHash, Transaction
    id for matching reference Number, which is used in future for managing
    certificate enrollment. Also update the corresponding status of the transaction
    with the IT-Admin/Console as applicable.*/
}
}
```

NOTE 2: List of applications that can access the certificate received from CA server. Behavior of access control for packages w.r.to different key stores are listed below.

- ✓ For **CCM** Keystore -- List can contain string "ALL" for all the packages, "WIFI" for wifi and Individual package names for each package.
- ✓ For **ANDROID** keystore -- List can contain string "ALL" for all the packages, "WIFI" for wifi.Currently there is no specific access control for each package except WIFI in Android Keystore.

3. Certificate Renewal

Renewal request sends hash of the certificate and list of packages which can use installed client certificate. Certificate hash is MD5 hash of the client certificate which needs to be renewed.

For renewal of the certificate, please follow below code snippet:

```
Public void testRenew(){
Context mContext = this.getApplicationContext();
EnterpriseKnoxManager ekm = null;
KnoxContainerManager kcm = null;
EnterpriseCertEnrollPolicy cep = null;
String mCEPProtocol = CEPConstants.CERT_PROFILE_TYPE_SCEP; // for SCEP Protocol
int mContainerId = 100;           // for container Id 100

//getting CEP policy object inside container
try {
    kcm = EnterpriseKnoxManager.getInstance()
                        .getKnoxContainerManager(mContext, mContainerId);
    cep = kcm.getEnterpriseCertEnrollPolicy(mCEPProtocol);
} catch (NoSuchMethodError e) {
      // TODO Auto-generated catch block
    Log.d(TAG,"Platform doesn't support CEP");
    e.printStackTrace();
}
//getting CEP policy outside the container i.e. inside user space(Owner)
/*
try {
    ekm = EnterpriseKnoxManager.getInstance();
    cep = ekm.getEnterpriseCertEnrollPolicy(mContext, mCEPProtocol);
} catch (NoSuchMethodError e) {
     // TODO Auto-generated catch block
    Log.d(TAG," Platform doesn't support CEP");
    e.printStackTrace();
}
*/

/* "regId" e.g. This is a unique number for each execution and will
basically match to the reference number (refNum) that is available/received
in broadcast receiver explained in NOTE 1. */
String regId = null;


/* certHash is the MD5 hash of the Client Certificate in hex format.
   Refer NOTE 3 for more details. */
String certHash = "13E26F6820EC0689B18A0915F3247D7A";

/* mList is list of All allowed packages. Please refer NOTE 2 for more
```

```
details.*/
List<String> mList = new ArrayList<String>();
// Email Application is allowed to access “cert”
mList.add(“com.android.email”);

//Trigger Certificate Renewal
if(cep != null) {
    regId = cep.renewUserCertificate(certHash, mList);
}
}
```

NOTE 3: “certHash” e.g. MD5 hash (in Hex format) of the client certificate that needs to be renewed. This is available through receiver, post enrollment for desired alias as explained in NOTE 1. Alternatively, agent can get the certificate directly from Android or CCM keystore using desired Alias and calculate the certificate hash using function similar to below one.

```
public static String getCertHash(String  alias) {
/* Agent/Caller can get the certificate using Keychain Apis given an
alias. It returns the X509Certificate chain for the requested alias, or
null if no there is no result.*/
X509Certificate certToHash =(X509Certificate)
                KeyChain.getCertificateChain(mContext, alias)[0];
certHash = getCertFingerprint(certToHash, “MD5”);
return certHash;
}

// Function to calculate the certificate MD5 hash in hex format.
public static String getCertFingerprint(X509Certificate certToHash,
String  hashAlgo) throws NoSuchAlgorithmException,
CertificateEncodingException
{
// TODO Auto-generated method stub
MessageDigest md = MessageDigest.getInstance(hashAlgo);
byte[] derEncoded = certToHash.getEncoded();
md.update(derEncoded);
byte[] hash = md.digest();
return byteToHexString(hash);
}
```

There are two input parameters:

- ✓ “hashAlgo” – "MD5"
- ✓ “certToHash” – X509Certificate for which we need to calculate hash.

4. Certificate Deletion

Certificate delete request sends hash of the certificate to be deleted. The return status shows if the certificate is deleted successfully or not. “0” refers to success otherwise failure.

For querying status of the certificate, please follow below code snippet:

```
Public void testDelete(){
Context mContext = this.getApplicationContext();
EnterpriseKnoxManager ekm = null;
KnoxContainerManager kcm = null;
EnterpriseCertEnrollPolicy cep = null;
```

```
String mCEPProtocol = CEPConstants.CERT_PROFILE_TYPE_SCEP; // for SCEP Protocol
int mContainerId = 100;           // for container Id 100

//getting CEP policy object inside container
try {
    kcm = EnterpriseKnoxManager.getInstance()
                        .getKnoxContainerManager(mContext, mContainerId);
    cep = kcm.getEnterpriseCertEnrollPolicy(mCEPProtocol);
} catch (NoSuchMethodError e) {
      // TODO Auto-generated catch block
    Log.d(TAG,"Platform doesn't support CEP");
    e.printStackTrace();
}
//getting CEP policy outside the container i.e. inside user space(Owner)
/*
try {
    ekm = EnterpriseKnoxManager.getInstance();
    cep = ekm.getEnterpriseCertEnrollPolicy(mContext, mCEPProtocol);
} catch (NoSuchMethodError e) {
     // TODO Auto-generated catch block
    Log.d(TAG," Platform doesn't support CEP");
    e.printStackTrace();
}
*/

//"regId" e.g. 0 for success otherwise failure error code.
int regId = null;

/* certHash is the MD5 hash of the Client Certificate in hex format.
   Refer NOTE 3 for more details.*/
String certHash = "13E26F6820EC0689B18A0915F3247D7A";

//Delete certificate
if(cep != null) {
    regId = cep. deleteUserCertificate(certHash);
}
}
```

5. Get Certificate transaction Status

Certificate status request sends transaction id of the certificate. It lets admin know if the certificate is installed or not.

For querying status of the certificate, please follow below code snippet:

```
Public void testgetStatus(){
Context mContext = this.getApplicationContext();
EnterpriseKnoxManager ekm = null;
KnoxContainerManager kcm = null;
EnterpriseCertEnrollPolicy cep = null;
String mCEPProtocol = CEPConstants.CERT_PROFILE_TYPE_SCEP; // for SCEP Protocol
int mContainerId = 100;           // for container Id 100


//getting CEP policy object inside container
```

```
try {
    kcm = EnterpriseKnoxManager.getInstance()
                        .getKnoxContainerManager(mContext, mContainerId);
    cep = kcm.getEnterpriseCertEnrollPolicy(mCEPProtocol);
} catch (NoSuchMethodError e) {
      // TODO Auto-generated catch block
    Log.d(TAG,"Platform doesn't support CEP");
    e.printStackTrace();
}

//getting CEP policy outside the container i.e. inside user space(Owner)
/*
try {
    ekm = EnterpriseKnoxManager.getInstance();
    cep = ekm.getEnterpriseCertEnrollPolicy(mContext, mCEPProtocol);
} catch (NoSuchMethodError e) {
     // TODO Auto-generated catch block
    Log.d(TAG," Platform doesn't support CEP");
    e.printStackTrace();
}
*/

//"regId" e.g. 0 for success otherwise failure error code.
int regId = null;
// transaction id. For more details refer NOTE 4.
String transactionId = "1a2b3c4d";
//Get the status of the enrollment/renew operation.
if(cep != null) {
    regId = cep. getCertEnrollmentStatus(transactionId);
}
}
```

NOTE 4: "transactionId" e.g. ID of the transaction for which status should be queried. This is available post enrollment/renewal operation through receiver as explained in section NOTE 1.