HOW TO WRITE AN NDES POLICY MODULE

1 Introduction

Prior to Windows Server 2012 R2, the Active Directory Certificate Services (ADCS) Network Device Enrollment Service (NDES) only supported certificate enrollment from within a trusted network. In order to extend NDES certificate enrollment to untrusted networks in Windows Server 2012 R2, NDES defines two new HTTP operations, NDESGenerateChallenge and NDESGetCACertThumbprint, as well as the new INDESPolicy third-party policy module COM API. With a third-party INDESPolicy COM assembly deployed, NDES on Windows Server 2012 R2 will be able to authenticate and authorize SCEP certificate requests over an untrusted network.

1.1 NDES HTTP Operations

After the INDESPolicy COM third-party custom policy module plug-in has been registered on a supporting NDES server installation (See Section 2 Registration), the following new HTTP operations will be exposed by the NDES server. Both of these new HTTP operations will be performed by the requesting party, typically the MDM solution, over a trusted HTTP(S) network.

1. NDESGenerateChallenge (Generating a challenge):

   The requesting party will call the new NDESGenerateChallenge HTTP operation over a trusted HTTP(S) connection to retrieve a password string, which NDES will generate by invoking INDESPolicy::GenerateChallenge on the registered policy module. The requesting party SHOULD forward the returned password string to a SCEP-compliant device in a secure manner, where it may be used for certificate enrollments (SCEP PkcsReq operations) over an untrusted network as the PKCS#9 challengePassword attribute within the embedded PKCS#10 certificate request.

2. NDESGetCACertThumbprint (Retrieving the CA certificate thumbprint):

   The requesting party will call the NDESGetCACertThumbprint HTTP operation over a trusted HTTP(S) connection to retrieve the MD5 thumbprint of the trust anchor for the CA targeted by the NDES server. This MD5 thumbprint will then be forwarded in a secure manner to the SCEP device. This way, the device can establish a trust anchor for validating SCEP responses from the NDES server over an untrusted network. This operation does not invoke any INDESPolicy API functions.

1.2 NDES Policy Module

The NDES policy module is to be implemented by the third-party developer as a free-threaded Windows INDESPolicy COM application and deployed on the R2 NDES server. When an INDESPolicy policy module is configured at the NDES server, the NDES server engine will call the INDESPolicy methods to support the following tasks:

1. Generating a challenge:

   When the NDESGenerateChallenge HTTP operation is executed, NDES will invoke the INDESPolicy::GenerateChallenge API function to generate the PKCS#9 challengePassword attribute value.

2. Verifying a SCEP request:

Verifying a PKCS #10 certificate request, which may include the challenge-password generated from Step #1 (for a new SCEP enrollment request) or not, in which case the SCEP request may be verified by virtue of it being signed by a trusted certificate (for a SCEP renewal request). This scenario calls into the INDESPolicy::VerifyRequest API function.

3. Notifying on SCEP status updates:

   Notifying the plug-in on lifecycle changes to the certificate request. For example, the certificate request may be denied, pended, or approved. This scenario calls into the INDESPolicy::Notify API function.

No more than one INDESPolicy COM third-party policy module may be registered on the NDES server at any time.

1.2.1 API Reference

The INDESPolicy COM API inherits from IUnknown and MUST be implemented as a thread-safe, free-threaded COM component.

Minimum supported client: None supported.
Minimum supported server: Windows Server 2012 R2.
Header: Certpol.h (include Certsrv.h).
Library: Certidl.lib.

1.2.1.1 INITIALIZE

This will be called by NDES when the NDES ISAPI extension is being loaded by IIS.

1.2.1.2 GENERATECHALLENGE

This will be called by NDES upon receipt of a NDESGenerateChallenge operation, passing in the configured NDES template with other parameters.

1.2.1.3 VERIFYREQUEST

This will be called by NDES to authenticate and authorize the SCEP new and renewal requests.

1.2.1.4 NOTIFY

This will be called by NDES to notify the INDESPolicy plug-in of changes to the certificate request.

1.2.1.5 UNINITIALIZE

This will be called by NDES when the NDES ISAPI extension is being unloaded by IIS.

1.2.2 Interface Definition

```
//+--------------------------------------------------------------------------
// INDESPolicy interface & enums
```

```
//+---------------------------------------------------------------------------
// SCEP defined message numbers, for reference only:
 typedef enum
X509SCEPMessageType
{
    SCEPMessageUnknown       = -1,
    SCEPMessageCertResponse   =  3, // Response to certificate/CRL request
SCEPMessagePKCSRequest    = 19, // PKCS#10 certificate request
    SCEPMessageGetCertInitial = 20, // Certificate polling (manual enroll)
                             // Issuer/Subject + XactId + SenderNonce
    SCEPMessageGetCert          = 21, // Retrieve a certificate
                            // Issuer/Serial + XactId + SenderNonce

    SCEPMessageGetCRL           = 22, // Retrieve a CRL (not supported)
} X509SCEPMessageType;

typedef enum X509SCEPDisposition
{
    SCEPDispositionUnknown = -1,    // Invalid value
    SCEPDispositionSuccess = 0,
    SCEPDispositionFailure = 2,
    SCEPDispositionPending = 3,
} X509SCEPDisposition;

typedef enum X509SCEPFailInfo
{
    SCEPFailUnknown           = -1,// Invalid value
    SCEPFailBadAlgorithm      = 0, // Unrecognized/unsupported algorithm ident.
    SCEPFailBadMessageCheck = 1, // Integrity check failed
    SCEPFailBadRequest        = 2, // Transaction not permitted or supported
    SCEPFailBadTime           = 3, // The signingTime attribute from the PKCS#7
                           // authenticatedAttributes was not
                           // sufficiently close to the system time
    SCEPFailBadCertId         = 4, // No certificate could be identified
matching
                         // the provided criteria
} X509SCEPFailInfo;



// INDESPolicy interface
[
    object,
    uuid(13CA515D-431D-46CC-8C2E-1DA269BBD625),
    helpstring("NDES Policy Module Interface"),
    local
]
interface INDESPolicy : IUnknown
{
    import "oaidl.idl";

    // Initialize NDES policy module

    HRESULT Initialize();


    // Uninitialize NDES policy module

    HRESULT Uninitialize();


    // Performs policy decision on whether to issue a challengePassword to
the
    // SCEP client.
    //
    // Parameters:
```

```
    //
    //  pwszTemplate:   The template being requested for, as determined by
NDES.
    //
    //  pwszParams:    Parameters specific to the policy module implementation.
    //
    //  ppwszResponse   After the user has been authenticated and authorized,
will
    //                     contain the user's SCEP challengePassword. NDES will
free
    //                    this using LocalFree.

    HRESULT GenerateChallenge(
                [in, ref] PCWSTR pwszTemplate,
                [in, ref] PCWSTR pwszParams,
                [out, retval] PWSTR *ppwszResponse);


    // Verifies the NDES certificate request for submission to the certification
authority.
    //
    // Parameters:
    //
    //  pctbRequest:            The encoded PKCS#10 request.
    //
    //  pctbSigningCertEncoded: The valid signing certificate for a renewal
request.
    //
    //  pwszTemplate:           The template being requested for, as determined
by NDES.
    //
    //  pwszTransactionId:      The SCEP request transaction ID.
    //
    //  pfVerified:              Should be set to TRUE if the pwszChallenge is
successfully verified,
    //                          and FALSE otherwise.

    HRESULT VerifyRequest(
                [in, ref] CERTTRANSBLOB* pctbRequest,
                [in, ref] CERTTRANSBLOB* pctbSigningCertEncoded,
                [in, ref] PCWSTR pwszTemplate,
                [in, ref] PCWSTR pwszTransactionId,
                [out, retval] BOOL* pfVerified);


    // Notifies the plugin of the transaction status of the SCEP certificate
request.
    //
    // Parameters:
    //
    //  pwszChallenge           The user's authentication and authorization
```

```
SCEP
    //                          challengePassword.
    //
    //  pwszTransactionId:      The SCEP request transaction ID.
    //
    //  disposition:            The disposition of the transaction.
    //
    //  lastHResult:            The HRESULT of the last operation.
    //
    //  pctbIssuedCertEncoded:  The requested certificate, if issued.

    HRESULT Notify(
                [in, ref] PCWSTR pwszChallenge,
                [in, ref] PCWSTR pwszTransactionId,
[in] X509SCEPDisposition disposition,
                [in] LONG lastHResult,
                [in, ref] CERTTRANSBLOB* pctbIssuedCertEncoded);
}
```

2 Registration

After installation and configuration of the NDES server, the INDESPolicy plug-in assembly may be registered with NDES through the following steps:

1. Register the INDESPolicy policy module on the NDES machine using the Windows regsvr32.exe COM registration utility.
2. Configure the NDES service account with Launch and Activation permissions on the INDESPolicy COM server, by using existing Windows COM configuration tools such as dcomcnfg.exe or oleview.exe. The NDES service account is the Windows account under which the NDES IIS SCEP app pool operates.
3. Register the ProgID of the INDESPolicy COM server with NDES by setting the following new registry value:

   [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\MSCEP\Modules]

   "**Policy**"="*name1.name2.version*"
4. Restart the NDES IIS SCEP app pool.

No more than one INDESPolicy COM third-party policy module may be registered on the NDES server at any time.

3 De-registration

The INDESPolicy plug-in assembly may be de-registered from NDES through the following steps:

1. De-register the ProgID of the INDESPolicy policy module with NDES by deleting the following registry value:

   [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\MSCEP\Modules]

   "**Policy**"="*name1.name2.version*"
2. De-register the INDESPolicy COM server on the NDES machine using the Windows regsvr32.exe COM registration utility.
3. Restart the NDES IIS SCEP app pool.

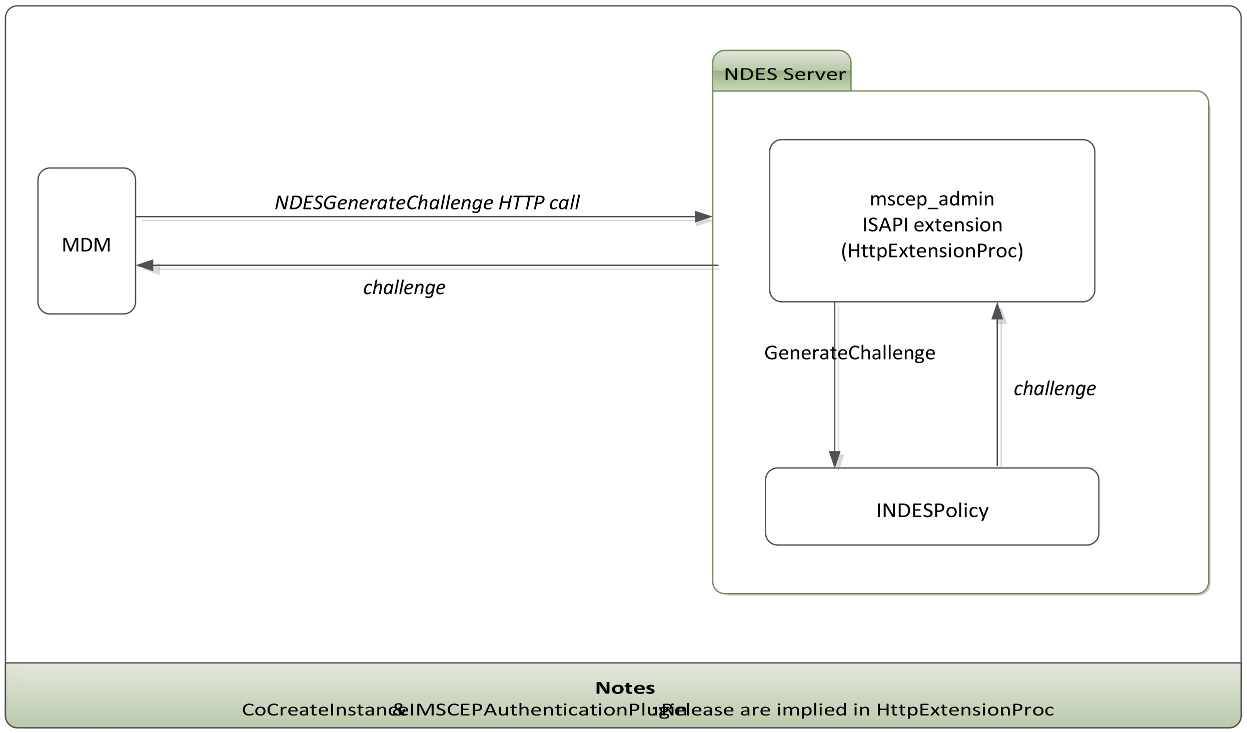
4 Scenarios

4.1 Generating a Challenge

NDES, implemented as an IIS ISAPI extension, currently exposes two IIS web application endpoints: mscep_admin for password generation, and mscep for SCEP operation support.

In this scenario, the mscep_admin endpoint of an NDES with a third-party INDESPolicy API policy module plug-in configured will be queried with the NDESGenerateChallenge operation over a trusted HTTP(S) connnection:

http://<server>/certsrv/mscep_admin[.dll]?**operation**=NDESGenerateChallenge&**keyUsage**=<keyUsageValue>&**params**=<pluginParameters>


NDES Server
MDM
NDESGenerateChallenge HTTP call
challenge
mscep_admin
ISAPI extension
(HttpExtensionProc)
GenerateChallenge
challenge
INDESPolicy
Notes
CoCreateInstance & IMSCEPAuthenticationPlugin::Release are implied in HttpExtensionProc

4.1.1 NDESGenerateChallenge Parameters

NDESGenerateChallenge accepts the operation, keyUsage,and params parameter values. Each parameter is caseinsensitive, and required (but, in the case of the Params parameter,may be set to an empty value). The parameters MUST also be provided in the expected order: operation, keyUsage, then params.

1. Operation

   This case-insensitive value identifies the operation being performed, and should be set to "NDESGenerateChallenge". This parameter is required.

2. KeyUsage

   This identifies the intended key usage value for the certificate and will be used by NDES to determine the certificate template to be requested from the issuing certification authority (CA). This parameter is required, and MUST be set to an unsigned decimal or hexagonal integer value (with preceding "0x" for a hexagonal value). Valid examples include 0xA0 & 160. The maximum integer value is (2^32 – 1) in decimal or hexagonal form. Note that the keyUsage parameter MUST precede the params parameter in the URL query string but come after the operation value.

3. Params

   This parameter is a string value opaque defined by the third-party developer and opaque to NDES. It is required, but may be set to an empty value, if the third-party developer of the policy module does not define any custom parameters. If non-empty, it MUST contain the policy-module custom parameters in valid C/C++ string form (that is, not containing the '\0' character). Valid custom data could be proprietary XML data, or base64url-encoded data.

Valid sample NDESGenerateChallenge queries could be:
https://server/certsrv/mscep_admin?**operation**=NDESGenerateChallenge&**keyUsage**=0xA0&**params**=

https://server/certsrv/mscep_admin?**operation**=NDESGenerateChallenge&**keyUsage**=0x80&**params**=MyStringData

4.1.2 NDESGenerateChallenge Processing

Upon receipt of a well-formed NDESGenerateChallenge operation, NDES will perform the following processing steps:

1. Determine the certificate template value from the keyUsage HTTP URL parameter as determined by the NDES EncryptionTemplate, SignatureTemplate, and GeneralPurposeTemplate configuration.
2. Invoke INDESPolicy::GenerateChallenge on the registered policy module, passing in the determined template and Params values.

4.1.3 NDESGenerateChallenge Return Value

The NDESGenerateChallenge HTTP operation will return the response from the INDESPolicy::GenerateChallenge API call (the value referred to by the ppwszResponse out parameter), which must be a non-NULL value.

4.1.4 INDESPolicy::GenerateChallenge Arguments

pwszTemplate: Set to the name of the registered NDES template corresponding to the NDESGenerateChallenge keyUsage parameter on the NDES server.

pwszParams: Set to the NDESGenerateChallenge params parameter value.

```
HRESULT
CNDESSamplePolicy::GenerateChallenge(
            /* [ref][in] */ const WCHAR *pwszTemplate,
            /* [ref][in] */ const WCHAR *pwszParams,
            /* [retval][out] */ WCHAR **ppwszResponse)

   HRESULT hr = S_OK;

    // Generate PKCS#9 challengePassword string
    // and set to *ppwszResponse
//error:
```

4.1.5 INDESPolicy::GenerateChallenge C++ Sample Code

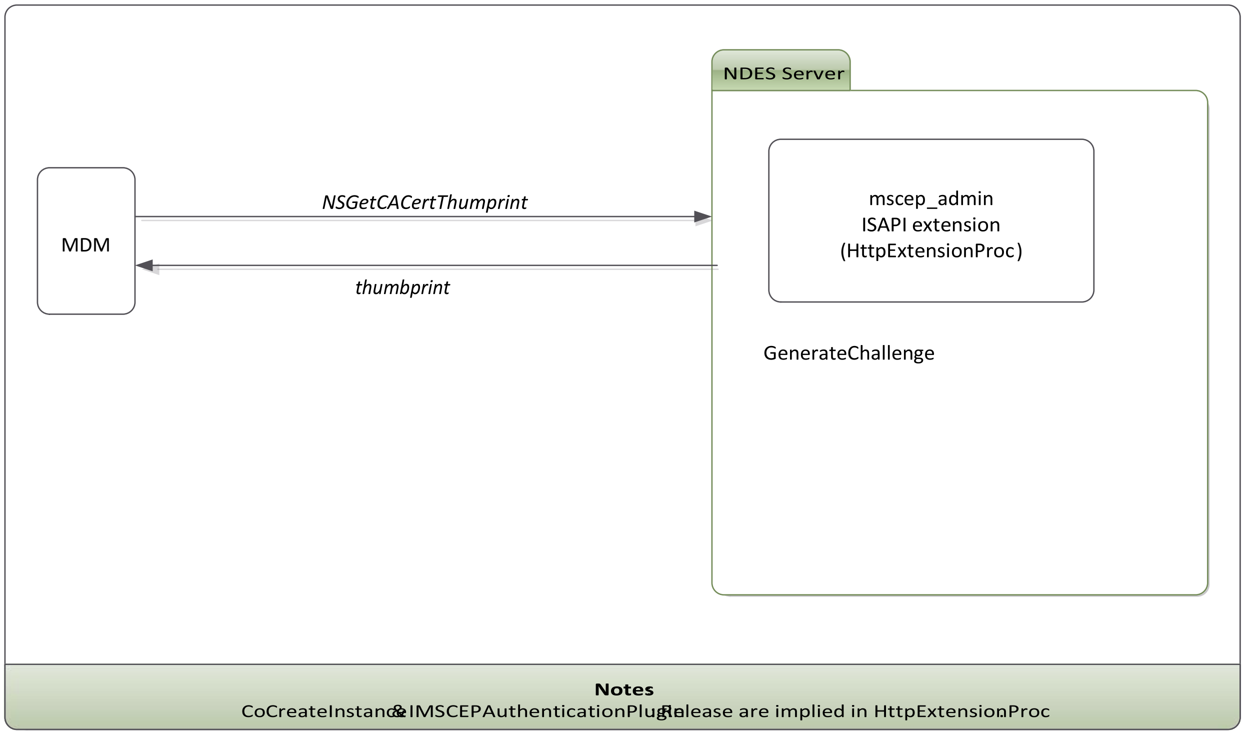
```
{

    return hr;

}
```

4.2 Retrieving the CA certificate thumbprint

In this scenario, the mscep_admin endpoint of an NDES with a third-party INDESPolicy API policy module plug-in configured will be queried with the NDESGetCACertThumbprint operation over a trusted HTTP(S) connnection:

http://<server>/certsrv/mscep_admin[.dll]?**operation**=NDESGetCACertThumbprint

This operation will NOT invoke the INDESPolicy plug-in.



4.2.1 NDESGetCACertThumbprint Parameters

1. Operation

This case-insensitive value identifies the operation being performed, and should be set to "NDESGetCACertThumbprint". This parameter is required.

4.2.2 NDESGetCACertThumbprint Processing

Upon receipt of a well-formed NDESGetCACertThumbprint operation, NDES will return the MD5 hash of the trust anchor for the NDES Registration Authority (RA) certificates as a hexadecimal string.

4.2.3 NDESGetCACertThumbprint Return Value

The NDESGetCACertThumbprint HTTP operation will return the MD5 hash of the trust anchor for the NDES Registration Authority (RA) certificates as a hexadecimal string.

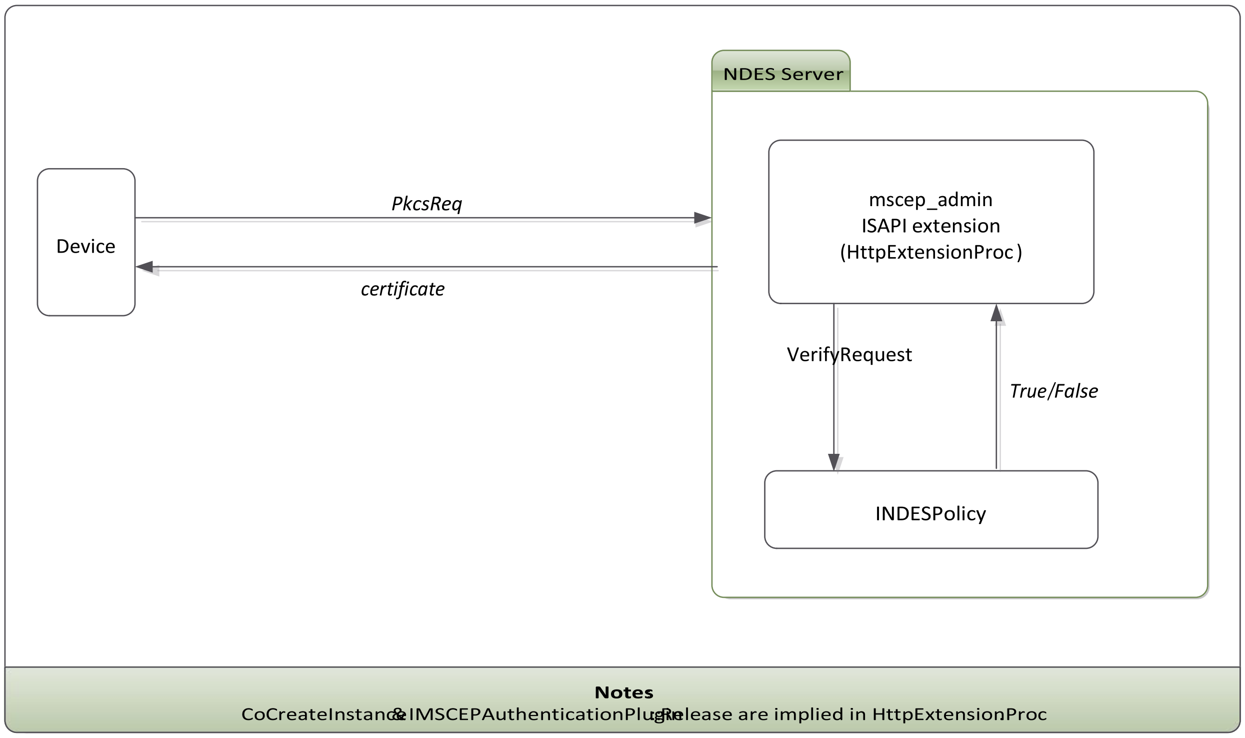
4.3 Verifying a SCEP Request

When NDES receives a SCEP PkcsReq certificate request (containing a PKCS#10 payload), NDES checks the PKCS#10 blob for an encoded PKCS#9 challengePassword attribute and classifies the request into one of the following:

1. A new PKCS#10 request:

   This contains a non-null PKCS#9 challengePassword attribute.
2. A renewal PKCS#10 request:

Among other things, this does not contain a PKCS#9 challengePassword attribute but was signed by a certificate that is trusted by the NDES server.



4.3.1 New request: INDESPolicy::VerifyRequest Arguments

pctbRequest: Set to the encoded PKCS#10 request blob, which contains the PKCS #9 challenge password.

pctbSigningCertEncoded: Set to NULL.

pwszTemplate: Set to the name of the configured NDES template matching the keyUsage value in the PKCS#10 request.

pwszTransactionId: Set to the SCEP request transaction ID.

4.3.2 Renewal request: INDESPolicy::VerifyRequest Arguments

pctbRequest: Set to the encoded PKCS#10 request blob.

pctbSigningCertEncoded: Set to the trusted, verified PKCS#10 signing certificate.

pwszTemplate: Set to the name of the configured NDES template matching the keyUsage value in the PKCS#10 request.

pwszTransactionId: Set to the SCEP request transaction ID.

4.3.3 INDESPolicy::VerifyRequest C++ Sample Code

```
HRESULT
CNDESSamplePolicy::VerifyRequest(
            /* [ref][in] */ CERTTRANSBLOB *pctbRequest,
            /* [ref][in] */ CERTTRANSBLOB *pctbSigningCertEncoded,
            /* [ref][in] */ const WCHAR *pwszTemplate,
            /* [ref][in] */ const WCHAR *pwszTransactionId,
            /* [retval][out] */ BOOL *pfVerified)
{
    HRESULT hr = S_OK;

    //to be freed
    IX509CertificateRequestPkcs10V3 *pP10Request = NULL;
    BSTR strEncodedRequest = NULL;
    PCCERT_CONTEXT pSigningCert = NULL;
    ICryptAttribute* pPwdCryptAtribute = NULL;
    PWSTR pwszPassword = NULL;

    *pfVerified = FALSE;
     if (NULL != pctbSigningCertEncoded
&&
        NULL != pctbSigningCertEncoded->pb &&
        0 != pctbSigningCertEncoded->cb)
    {
        // Since a signing certificate was passed through,
        // this is a renewal request.
        // The SCEP PkcsReq signing certificate has already been verified as
        // trusted by the NDES server.
        // You can verify the SCEP PkcsReq request based solely on this
signing
        // certificate, OR, additionally, based on a PKCS#9 challengePassword
attribute
        // (this is not required by the SCEP protocol for renewal requests).
         pSigningCert = CertCreateCertificateContext(
X509_ASN_ENCODING,
pctbSigningCertEncoded->pb,
pctbSigningCertEncoded->cb);

        if (NULL == pSigningCert)
        {
            hr = E_OUTOFMEMORY;
goto error;
        }

        // Additional validation on the signing certificate
    }
else
    {
        // Since there is no trusted signing certificate passed through,
        // this is a brand new enrollment request. The SCEP PkcsReq request
// was signed by a dummy certificate (not passed through) and should be
validated by
        // its PKCS#9 challengePassword attribute.

        //1: Load up PKCS#10 request
        strEncodedRequest = SysAllocStringByteLen((LPCSTR)pctbRequest->pb,
pctbRequest->cb);
        if (NULL == strEncodedRequest)
        {
            hr = E_OUTOFMEMORY;
            goto error;
        }

        hr = CoCreateInstance( CLSID_CX509CertificateRequestPkcs10,
                               NULL,
                               CLSCTX_INPROC_SERVER,
                               IID_IX509CertificateRequestPkcs10V3,
                               (void **)&pP10Request);
        if (FAILED(hr))
        {
            goto error;
        }

        hr = pP10Request->InitializeDecode(strEncodedRequest,
EncodingType::XCN_CRYPT_STRING_ANY);
        if (FAILED(hr))
        {
            goto error;
        }

        //2: Extract the PKCS#9 challengePassword attribute from the PKCS#10
request
        hr = GetCryptAttribute(pP10Request,
                        TEXT(szOID_RSA_challengePwd),
                        &pPwdCryptAtribute);
        if (FAILED(hr))
        {
            goto error;
        }

        if (NULL == pPwdCryptAtribute)
        {
            //no password: disallow
            hr = E_INVALIDARG;
            goto error;
        }

        hr = GetStringValueAtIndex(
                    pPwdCryptAtribute,
                    0,
                    &pwszPassword);
        if (FAILED(hr))
        {
            goto error;
        }

        if (NULL == pwszPassword)
        {
            //empty password: disallow
            hr = E_INVALIDARG;
            goto error;
        }

        //3: Left to do: Verify PKCS#9 challengePassword attribute & set
*pfVerified
```

```
}
```

```
error:
    LocalFree(pwszPassword);
    LocalFree(pPwdCryptAtribute);
    if (NULL != pSigningCert)
    {
        CertFreeCertificateContext(pSigningCert);
    }
    if (NULL != strEncodedRequest)
```

```
        }
        hr = pCurrCryptAttrib->get_ObjectId(&pOid);
        if (FAILED(hr))
        {
            goto error;
        }
```

```
        if (FAILED(hr))
        {
            goto error;
        }

        pOid->get_Value(&strOid);
        if (FAILED(hr))
        {
```

```
        goto error;
}
```

```
    hr = pX509Attributes->get_Count(&cAttributes);
```

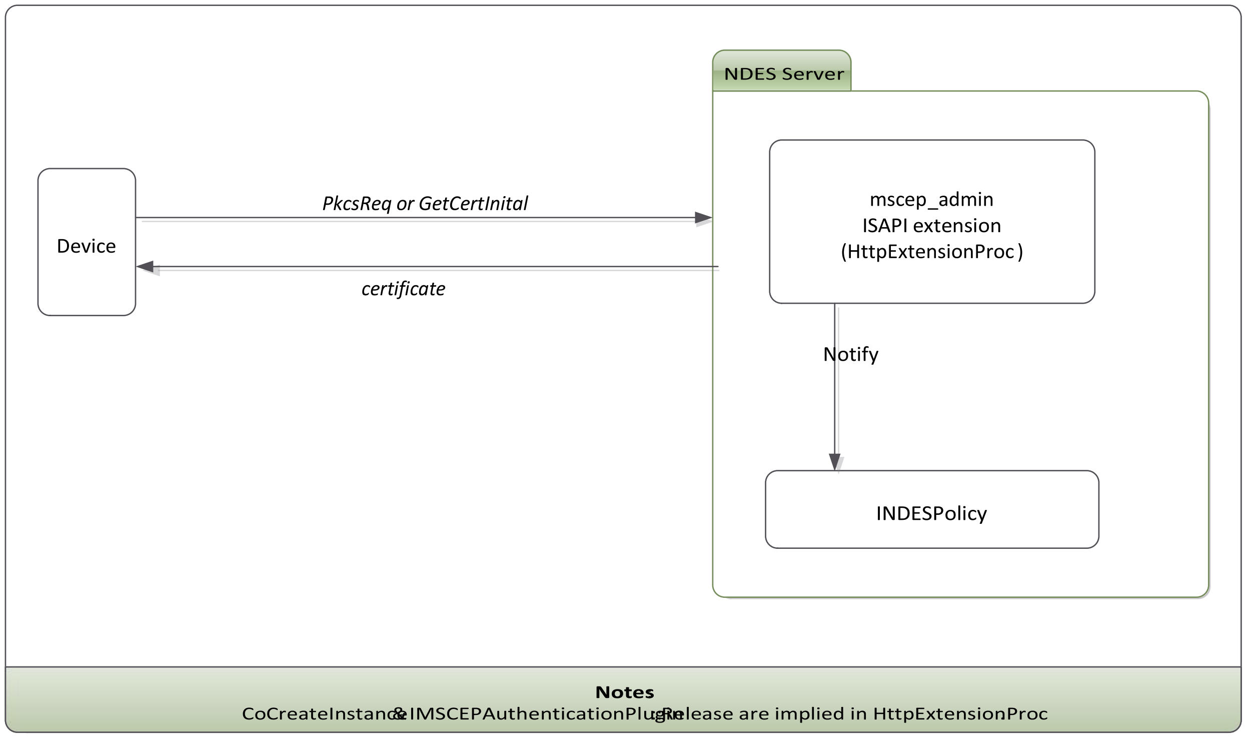
```
    hr = pX509Attributes->get_Count(&cAttributes);
    if (FAILED(hr))
    {
        goto error;
    }

    pX509Attributes->get_ItemByIndex(index, &pAttribute);
    if (FAILED(hr))
    return
hr; }
```

4.4 Notifying on SCEP Status Updates

NDES submits SCEP certificate enrollment requests to the targeted CA, which may then deny, pend, or approve the request, etc. NDES will call INDESPolicy::Notify in either of the following lifecycle scenarios:

1. After the status of the request is received from the CA, to notify the policy module plug-in of the SCEP PkcsReq request status.
2. After NDES queries the CA for an updated request status, because the client submitted a SCEP GetCertInitial request.

In both cases, the INDESPolicy::Notify API function will be invoked by NDES asynchronously on a thread separate from the SCEP request processing thread.



4.4.1 Notifying INDESPolicy::Notify Arguments

pwszChallenge: Set to the PKCS#9 challengePassword in the PKCS#10 request, if available.

pwszTransactionId: Set to the SCEP request transaction ID.

disposition: Set to the disposition of the transaction. lastHResult:
The HRESULT returned from the last NDES operation.

pctbIssuedCertEncoded: Set to the requested certificate, if issued (otherwise NULL).

4.4.2 INDESPolicy::Notify C++ Sample Code

```
HRESULT
CNDESSamplePolicy::Notify(
            /* [ref][in] */ const WCHAR *pwszChallenge,
            /* [ref][in] */ const WCHAR *pwszTransactionId,
            /* [in] */ X509SCEPDisposition disposition,
            /* [in] */ LONG lastHResult,
            /* [ref][in] */ CERTTRANSBLOB *pctbIssuedCertEncoded)
{
    HRESULT hr = S_OK;

    // Invoked asynchronronously
switch (disposition)
    {           case
SCEPDispositionSuccess:
        // pctbIssuedCertEncoded will contain an issued certificate
break;
         case
SCEPDispositionFailure:
break;
         case
SCEPDispositionPending:
break;
         case
SCEPDispositionUnknown:
break;

default:

    }
//error:

    return hr;

break;




}
```