ComponentSpace SAML for ASP.NET Certificate Guide

Contents

Introduction

X.509 certificates are used to secure SAML SSO between the identity provider and service provider.

Refer to the SAML Primer for an overview of the security considerations.

This guide describes the generation, management, and configuration of X.509 certificates used to secure SAML SSO.

Transport Layer Security Certificates

The SAML specification recommends that all communications are over HTTPS.

As the majority of use cases see the SAML messages exchanged between the identity provider and service provider via the browser, it's important to ensure certificates for HTTPS are issued by a certificate authority (CA).

Certificates not issued by a CA (eg. self-signed certificates) will result in the browser presenting a warning message to the user.

XML Signature and Encryption Certificates

Certificates used for XML signature and/or XML encryption support may be:

- Self-signed
- CA-issued
- Shared with HTTPS

The best option will depend on the specific business requirements.

Potential advantages and disadvantages are outlined in the following sections.

Self-Signed Certificates

Self-signed certificates have the following advantages:

- No cost
- May be created as required
- May have longer expiry times than CA-issued certificates

They have the following disadvantages:

- Certificate chain cannot be validated as the certificate of the issuer is the same certificate

Although self-signed certificates cannot be validated, their use will be limited to a number of partner providers. A self-signed certificate securely received from a partner provider may be trusted as it's received from a known source.

CA-Issued Certificates

CA-issued certificates have the following advantages:

- Certificate chain can be validated
- Support for certificate revocation lists (CRLs)

They have the following disadvantages:

- Cost
- May be a delay in issuance

Certificate Validation

Validation of certificate chains including checking CRLs can be a relatively expensive operation, especially if it requires off-server communications to retrieve CRLs.

The SAML component doesn't perform this validation of certificates. Instead, this is left to the application as it will vary with business requirements.

It's recommended that certificate validation not occur as part of SAML SSO. Instead, if required, it should be considered a separate operation that's performed on a regular basis (e.g. nightly). A revoked certificate would require re-issuance of the certificate and coordination with partner providers using the certificate.

HTTPS Shared Certificates

CA-issued certificates for SAML HTTPS endpoints also may be used for XML signature and/or XML encryption.

Sharing certificates has the following advantages:

- All the advantages of CA-issued certificates
- More cost effective

Sharing has the following disadvantages:

- All the disadvantages of CA-issued certificates
- If the certificate is compromised, security is compromised at both the transport and application layer

Certificate Storage

SAML configuration supports certificates stored in:

- Certificate file
- Windows certificate store
- Certificate string
- Application configuration
- Azure key vault

The best option will depend on the specific business requirements.

Certificate Files

Certificates may be stored on the file system as base-64 encoded or DER encoded .CER files.

A certificate and its associated private key may be stored on the file system as a .PFX file.

These are the certificate file formats supported by Windows and the .NET framework.

A local provider certificate stored on the file system may be specified in the SAML configuration.

```
<LocalCertificates>
```

```
  <Certificate FileName="Certificates\sp.pfx" Password="password"/>
</LocalCertificates>
```

A local provider certificate file always will be a .PFX as it must include the private key. The password protects the .PFX file.

A partner provider certificate stored on the file system may be specified in the SAML configuration.

```
<PartnerCertificates>
  <Certificate FileName="Certificates\idp.cer"/>
</PartnerCertificates>
```

A partner provider certificate file always will be a .CER as it contains the public key only. A password is not required to protect the .CER file.

Windows Certificate Store

Certificates and their associated private keys, if any, may be stored in the Windows certificate store.

A local provider certificate stored in the Windows certificate store may be specified in the SAML configuration. The certificate must include a private key.

Refer to the Private Key Permissions section to ensure the application process has read permission for the private key.

The certificate may be identified by its serial number.

```
<LocalCertificates>
  <Certificate SerialNumber="74f0ebfe22358db8433138f9558c9af9"/>
</LocalCertificates>
```

Alternatively, the certificate may be identified by its thumbprint.

```
<LocalCertificates>
  <Certificate Thumbprint="a6a4ae4e0b378ec73678e5812690af50e3ec3769"/>
</LocalCertificates>
```

Or the certificate may be identified by its subject name.

```
<LocalCertificates>
  <Certificate SubjectName="www.idp.com"/>
</LocalCertificates>
```

Similarly, a partner provider certificate stored in the Windows certificate store may be specified in the SAML configuration. The certificate will not include a private key.

The certificate may be identified by its serial number, thumbprint or subject name.

```
<PartnerCertificates>
 <Certificate SerialNumber="0867a17dc9efeabe4ccbf7e7adb7c37a "/>
</PartnerCertificates>
```

Store Location and Name

By default, certificates are expected to be stored in the local machine's personal certificate store.

In a hosted environment, instead of the local machine's store, the current user store may be used.

```
<LocalCertificates>
 <Certificate
  StoreLocation="CurrentUser"
  SerialNumber="74f0ebfe22358db8433138f9558c9af9"/>
</LocalCertificates>
```

Generally, it's recommended that certificates are stored in the personal certificate store. However, it is possible to reference certificates stored elsewhere.

```
<LocalCertificates>
 <Certificate
  StoreName="WebHosting"
  SerialNumber="74f0ebfe22358db8433138f9558c9af9"/>
</LocalCertificates>
```

Certificate Strings

Certificates may be specified as base-64 encoded strings.

This facilitates storing certificates in a database and setting SAML configuration programmatically.

A local provider certificate string may be specified in the SAML configuration.

```
<LocalCertificates>
 <Certificate String="MIIC/jCCAeagAwIBAgIQ..." Password="password"/>
</LocalCertificates>
```

A local provider certificate string is the base-64 encoded bytes making up the certificate and its private key. The password protects the certificate string.

PowerShell may be used to convert a PFX certificate file to a base-64 string.

For example:

```
$bytes = [System.IO.File]::ReadAllBytes("idp.pfx")
[System.Convert]::ToBase64String($bytes)
```

Alternatively, the Microsoft utility, CertUtil, may be used to convert a PFX certificate file to base-64.

For example:

```
Certutil.exe -encode c:\certs\idp.pfx c:\certs\b64-idp.pfx
```

A partner provider certificate string may be specified in the SAML configuration.

```
<PartnerCertificates>
  <Certificate String="MIIDATCCAemgAwIBAgIQ..."/>
</PartnerCertificates>
```

A partner provider certificate string contains the public key only. A password is not required to protect the certificate string.

The Microsoft utility, CertUtil, may be used to convert DER-encoded certificate files to base-64.

For example:

```
Certutil.exe -encode c:\certs\idp.cer c:\certs\b64-idp.cer
```

Application Configuration

Certificates may be stored as part of the application's configuration.

These certificates are accessed through the System.Configuration.ConfigurationManager.AppSettings class and identified by configuration keys.

An optional password may be specified if required.

The method for setting this configuration is left to the application. However, one use is to access certificates stored in an Azure key vault. Refer to the Azure Key Vault section for more information.

The configuration value is the certificate, and optionally its private key, encoded as a base-64 string.

The Certificate Strings section describes how to convert certificates and private keys into base-64 encoded strings.

```
<LocalCertificates>
  <Certificate Key="IdP" Password="password"/>
</LocalCertificates>
```

Partner certificates may be similarly specified.

Azure Key Vault

Certificates may be stored in an Azure key vault.

For general information on the Azure key vault, refer to:

https://docs.microsoft.com/en-us/azure/key-vault/

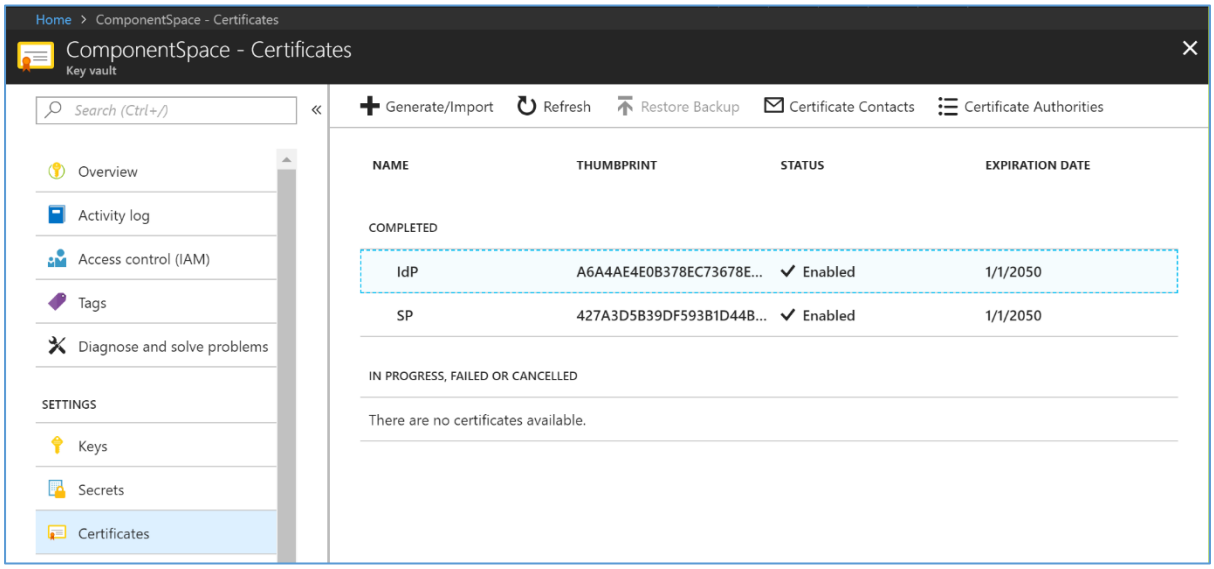
Note that only certificates with private keys may be stored in the key vault.

Also note that applications do not have to be deployed to Azure to take advantage of an Azure key vault.

The Azure configuration steps are:

- Register in Azure Active Directory the application that will access the key vault, including a password.
- Create the key vault and import or generate certificates.
- Set the key vault access policies to permit the registered application to get and list the keys, secrets and certificates.

The following screenshot shows two certificates, named IdP and SP, that have been added to the key vault.



The Name specified in the key vault is used as the SAML certificate configuration key.

A password is not required.

```
<LocalCertificates>
  <Certificate Key="IdP"/>
</LocalCertificates>
```

Key Vault Access

The application is responsible for establishing a connection to the key vault.

The following example code retrieves a certificate from the key vault and makes it available through the application settings. This code should be run at application start-up.

```
private KeyVaultClient GetKeyVaultClient(string clientId, string clientSecret)
{
  return new KeyVaultClient(
    new KeyVaultClient.AuthenticationCallback(async (string authority, string resource,
string scope) =>
    {
      var authenticationContext = new AuthenticationContext(authority,
TokenCache.DefaultShared);
      var clientCredential = new ClientCredential(clientId, clientSecret);
      var authResult = await authenticationContext.AcquireTokenAsync(resource,
clientCredential);

      return authResult.AccessToken;
    }));
}

private async Task<string> GetCertificateStringFromKeyVaultAsync(KeyVaultClient
keyVaultClient, string vaultUrl, string certificateName)
{
  var secret = await keyVaultClient.GetSecretAsync(vaultUrl, certificateName);

  return secret.Value;
}

// Save the certificate from the key vault in the application settings.
var keyVaultClient = GetKeyVaultClient(
  "60e201d2-e713-4551-b792-339b81fdae07",
  "Wn7mlKqhQcpIyDkZIu2+WlO8F8IIMFRXGHx94CYPZUo=");

ConfigurationManager.AppSettings["IdP"] = GetCertificateStringFromKeyVaultAsync(
  keyVaultClient,
  "https://componentspace.vault.azure.net/",
  "idp").Result;
```

The ClientId and ClientSecret correspond to the application ID and password generated at the time the application was registered in Azure Active Directory.

Certificate Use

By default, certificates may be used for both signature generation/verification and XML encryption/decryption. If required, separate certificates may be configured.

In the following example, separate certificates are used for decrypting SAML assertions and signing SAML messages at the service provider.

```
<LocalCertificates>
 <Certificate FileName="Certificates\sp-enc.pfx" Password="password" Use="Encryption"/>
 <Certificate FileName="Certificates\sp-sig.pfx" Password="password" Use="Signature"/>
</LocalCertificates>
```

Similarly, separate certificates are used for encrypting SAML assertions and verifying SAML message signatures at the identity provider.

```
<PartnerCertificates>
 <Certificate FileName="Certificates\sp-enc.cer" Use="Encryption"/>
 <Certificate FileName="Certificates\sp-sig.cer" Use="Signature"/>
</PartnerCertificates>
```

Certificate Rollover

Certificate rollover refers to replacing a certificate that's about to expire or that potentially has been compromised.

Local Certificates

If a local certificate is to be rolled over, this may be done simultaneously for all partner providers but this may present problems coordinating with each of the partner providers.

Instead, a new certificate may be introduced in a phased roll out. To achieve this, a local certificate can be specified as part of the partner provider configuration. This local certificate overrides any local certificate specified for the local provider.

In the following example, a local certificate is specified for the local identity provider.

This will be used to sign SAML responses or assertions sent to partner services providers, unless overridden by the partner service provider configuration.

The first partner service provider specifies idp1.pfx as the local certificate. This will be used to sign SAML responses or assertions sent to this partner service provider.

The second partner service provider specifies idp2.pfx as the local certificate. This will be used to sign SAML responses or assertions sent to this partner service provider.

The third service provider doesn't specify a local certificate. Therefore, idp.pfx, which is specified in the local identity provider configuration, will be used to sign SAML responses or assertions sent to this partner service provider.

```
<IdentityProvider
 Name="http://ExampleIdentityProvider">
 <LocalCertificates>
  <Certificate FileName="Certificates\idp.pfx" Password="password"/>
 </LocalCertificates>
</IdentityProvider>

<PartnerServiceProviders>
```

```
 <PartnerServiceProvider
  Name="http://ExampleServiceProvider1">
  <LocalCertificates>
   <Certificate FileName="Certificates\idp1.pfx" Password="password"/>
  </LocalCertificates>
  <PartnerCertificates>
   <Certificate FileName="Certificates\sp1.cer"/>
  </PartnerCertificates>
 </PartnerServiceProvider>

 <PartnerServiceProvider
  Name="http://ExampleServiceProvider2">
  <LocalCertificates>
   <Certificate FileName="Certificates\idp2.pfx" Password="password"/>
  </LocalCertificates>
  <PartnerCertificates>
   <Certificate FileName="Certificates\sp2.cer"/>
  </PartnerCertificates>
 </PartnerServiceProvider>

<PartnerServiceProviders>
 <PartnerServiceProvider
  Name="http://ExampleServiceProvider3">
  <PartnerCertificates>
   <Certificate FileName="Certificates\sp3.cer"/>
  </PartnerCertificates>
 </PartnerServiceProvider>
</PartnerServiceProviders>
```

To support phased rollout of a new certificate, it could be specified as the local certificate for the local provider and the old certificate is specified as the local certificate for each partner provider. As partner providers are ready to switch to the new certificate, the local certificate specifications for these partner providers are removed so that the new certificate is then used.

Partner Certificates

If a partner provider certificate is being rolled over, this may present problems coordinating with the partner provider.

To avoid any issues, both the old and new partner provider certificate may be specified.

In the following example, both the old and new certificates are specified for the partner provider.

If a signed SAML message is received from the partner provider, the first certificate is used to attempt to verify the signature. If that fails, the second certificate is used.

```
<PartnerServiceProvider
 Name="http://ExampleServiceProvider">
 <PartnerCertificates>
  <Certificate FileName="Certificates\sp-new.cer"/>
```

```
  <Certificate FileName="Certificates\sp-old.cer"/>
 </PartnerCertificates>
</PartnerServiceProvider>
```

Once successful rollover of the partner provider's certificate has been confirmed, the configuration should be updated to remove the old certificate.

Certificate File Formats

Windows and the .NET framework support a number of certificate file formats.

DER Format

The Distinguished Encoding Rules (DER) format stores X.509 certificates as binary.

Standard file extensions are .cer, .crt and, less commonly on Windows, .der.

PEM Format

The Privacy-enhanced Electronic Email (PEM) format stores X.509 certificates as base-64 encoded strings.

The certificate string is wrapped by "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----" although these are optional on Windows.

Standard file extensions are .cer, .crt and, less commonly on Windows, .pem.

PKCS#12 Format

Public-Key Cryptography Standards (PKCS) #12 stores X.509 certificates and associated private keys as binary.

Normally the content is protected by a password.

Standard file extensions are .pfx and, less commonly on Windows, .p12.

Certificate File Permissions

The account under which the application is running must have read permission to the certificate file.

For a .PFX file, the account also must have read permission to the location where the private key is stored.

Private keys are stored in containers on the file system. Typically, the location of the private key container is:

C:\ProgramData\Microsoft\Crypto\RSA\MachineKeys

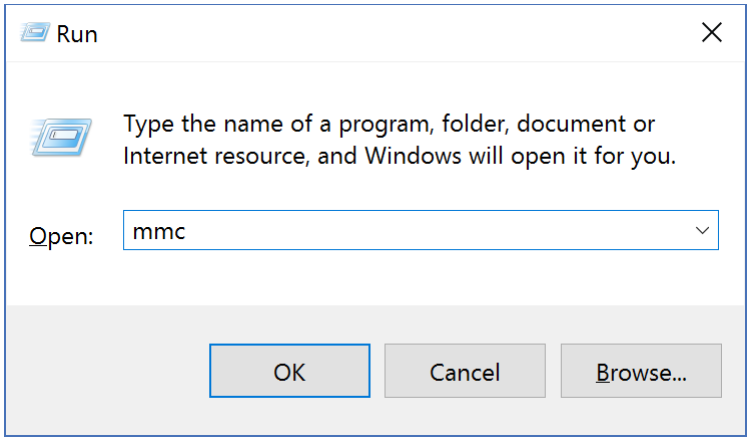
If required, the FindPrivateKey.exe Windows SDK utility may be run to locate the private key container.

http://msdn.microsoft.com/en-us/library/aa717039.aspx

The application account must have permission to create a file in the private key container folder.

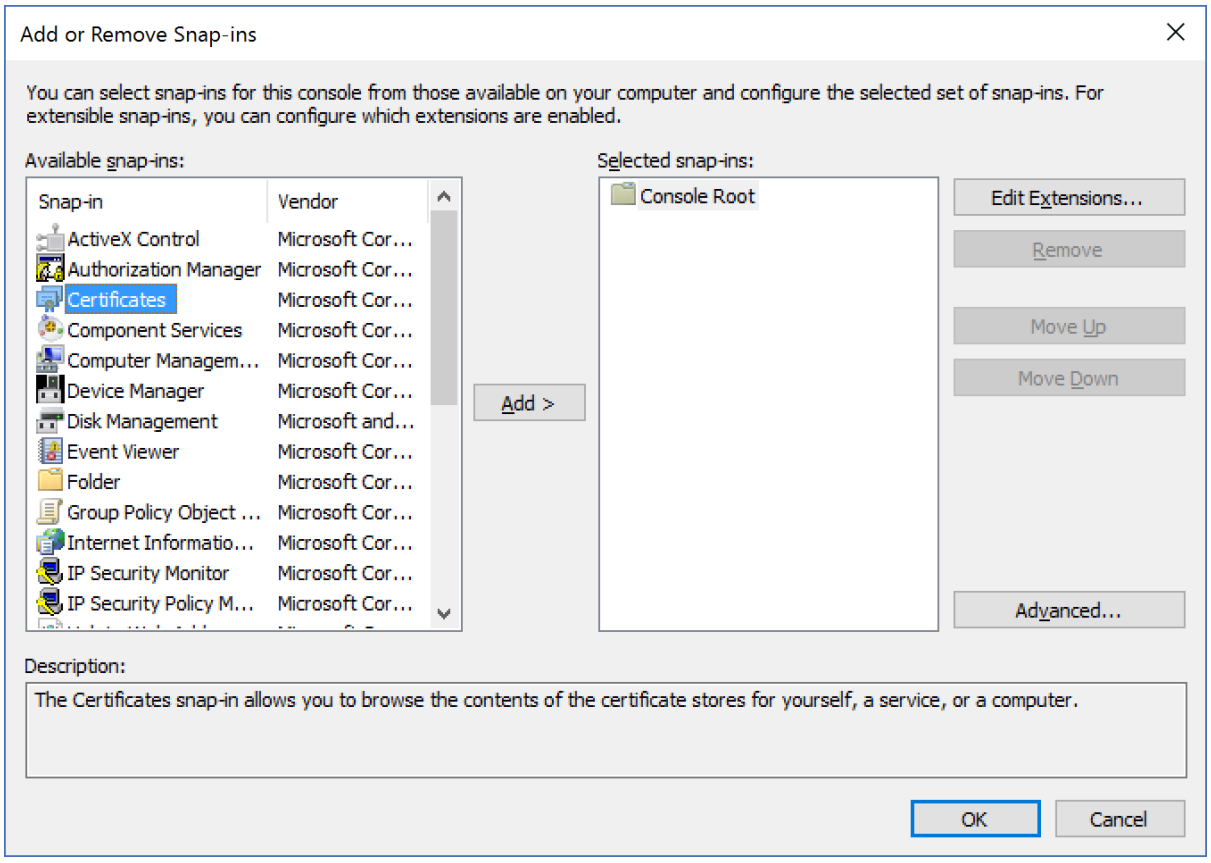
Using the MMC Certificates Snap-In

The Microsoft Management Console (MMC) Certificates snap-in may be used to manage the Windows certificate store.

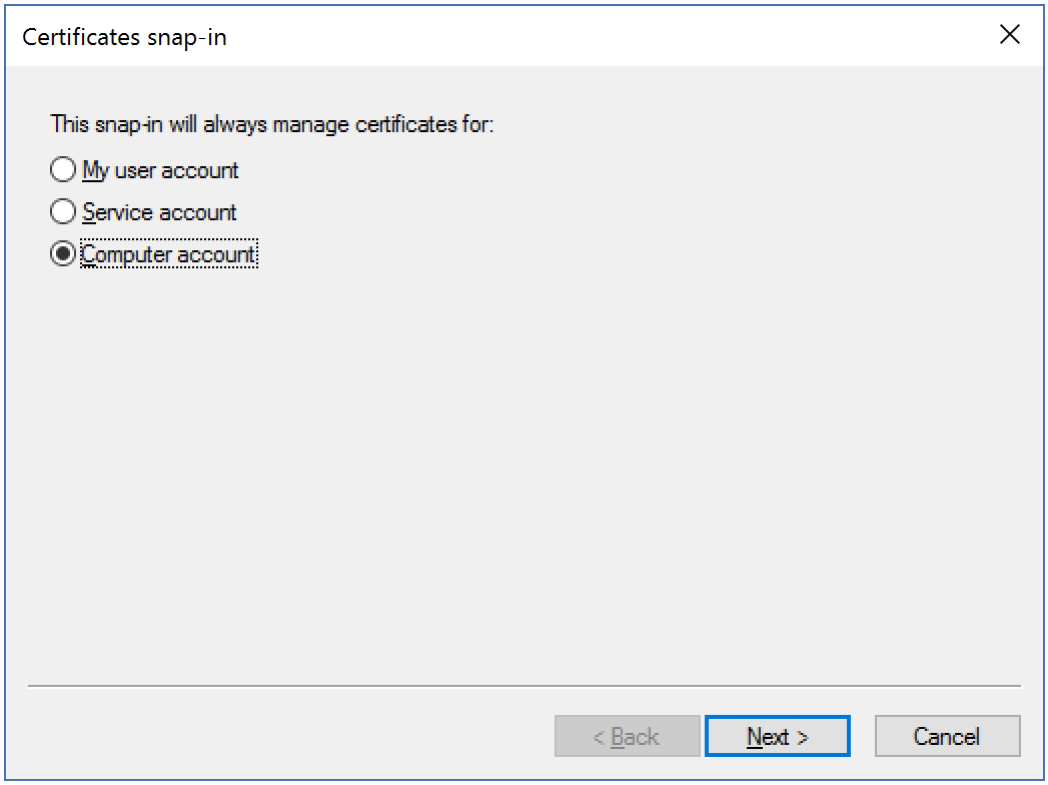


From the main menu, select File > Add/Remove Snap-in.

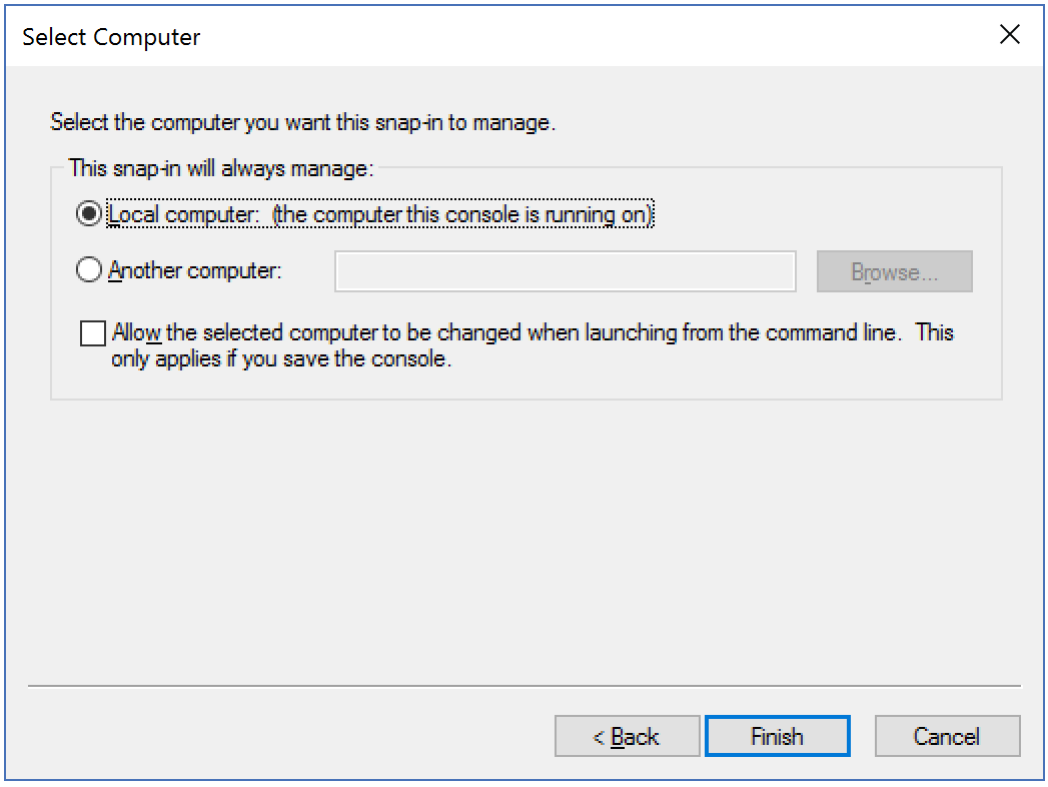
Select the Certificates snap-in and click the Add button.



Specify the computer account. This corresponds to the local machine store location.

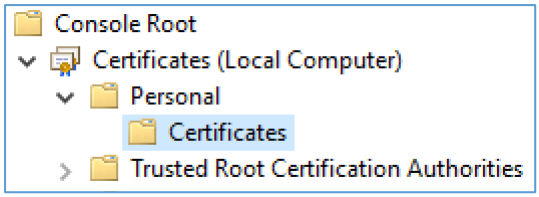


Select the local computer.

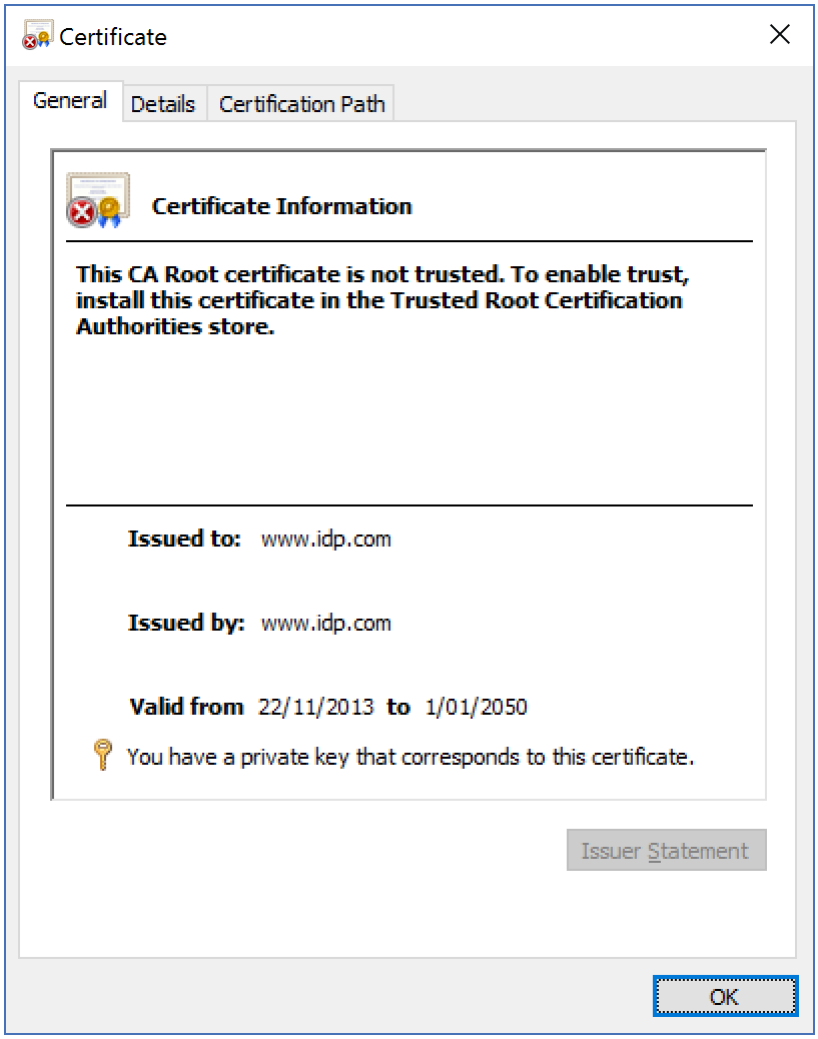


Click the Finish button and then click OK to dismiss the resulting dialog.

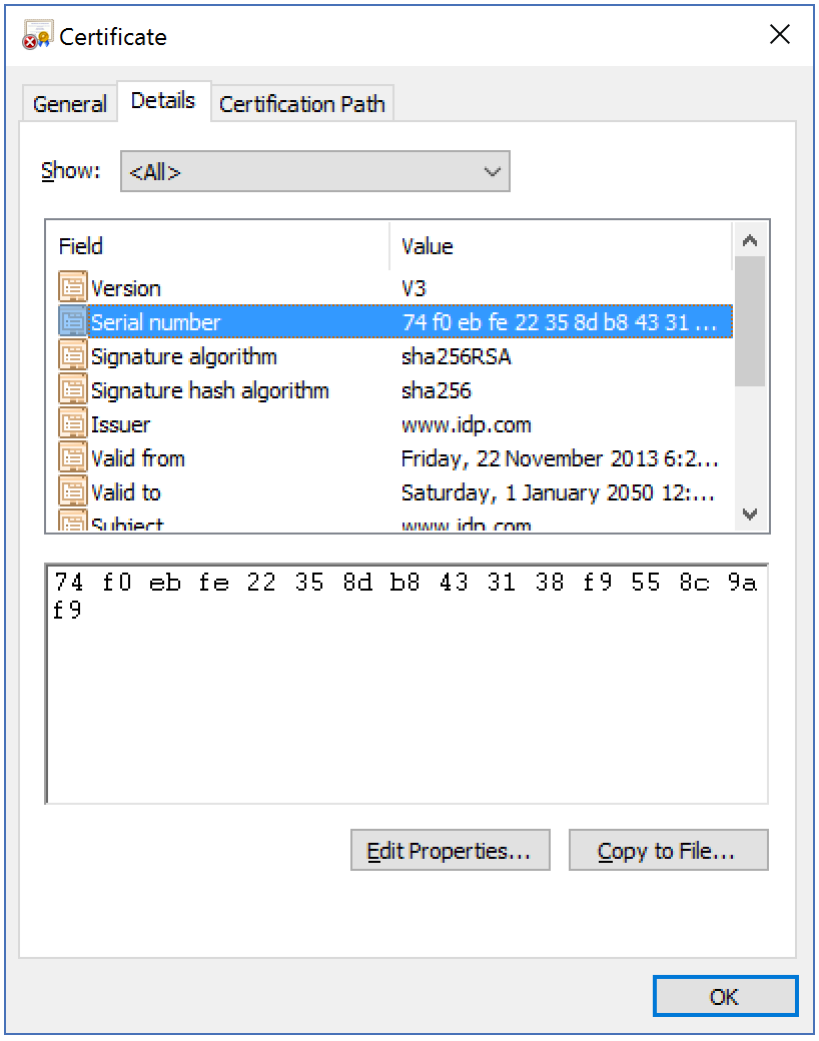
Expand the tree view to show the personal certificates.



Double-click on a certificate to open it.

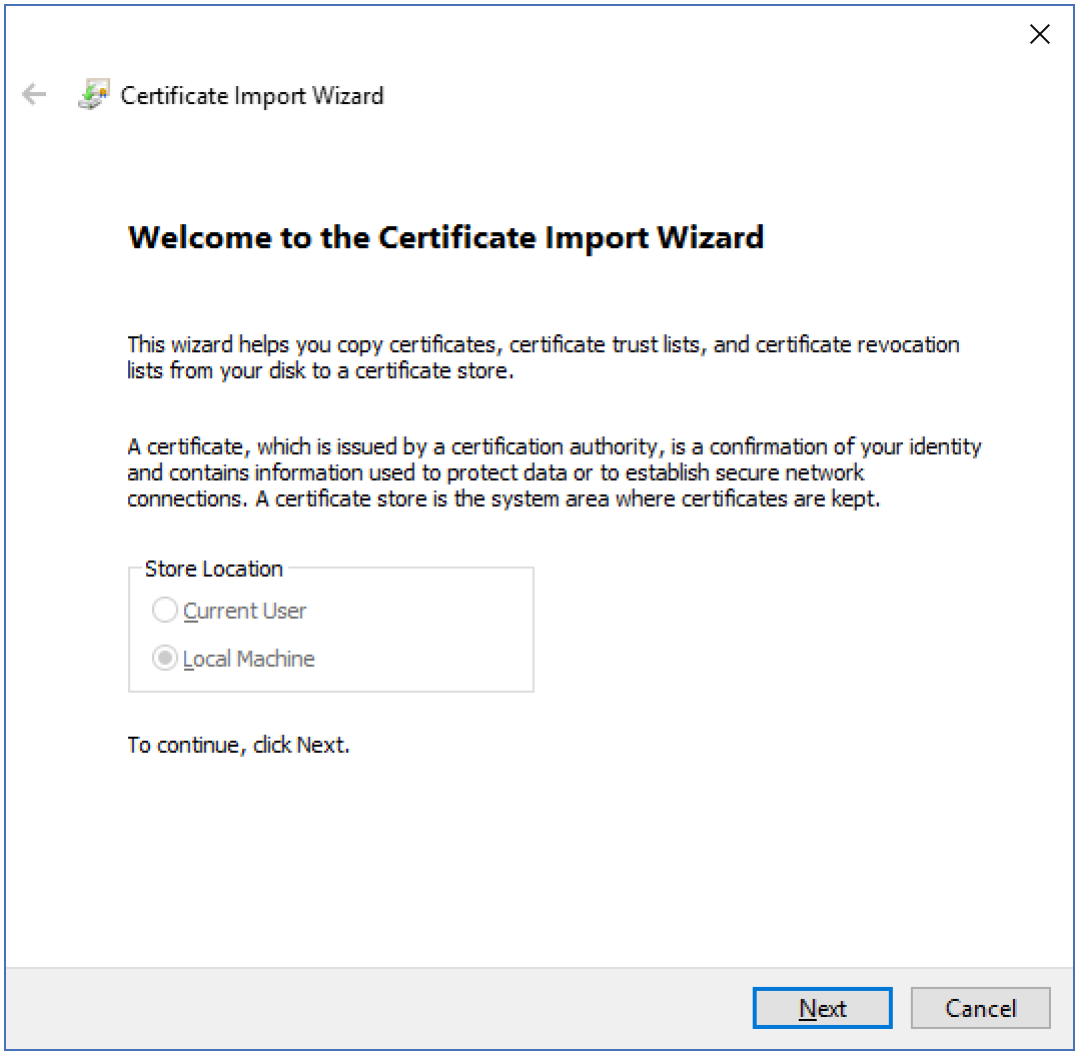


The details tab includes information such as the certificate's serial number, thumbprint, and subject name.

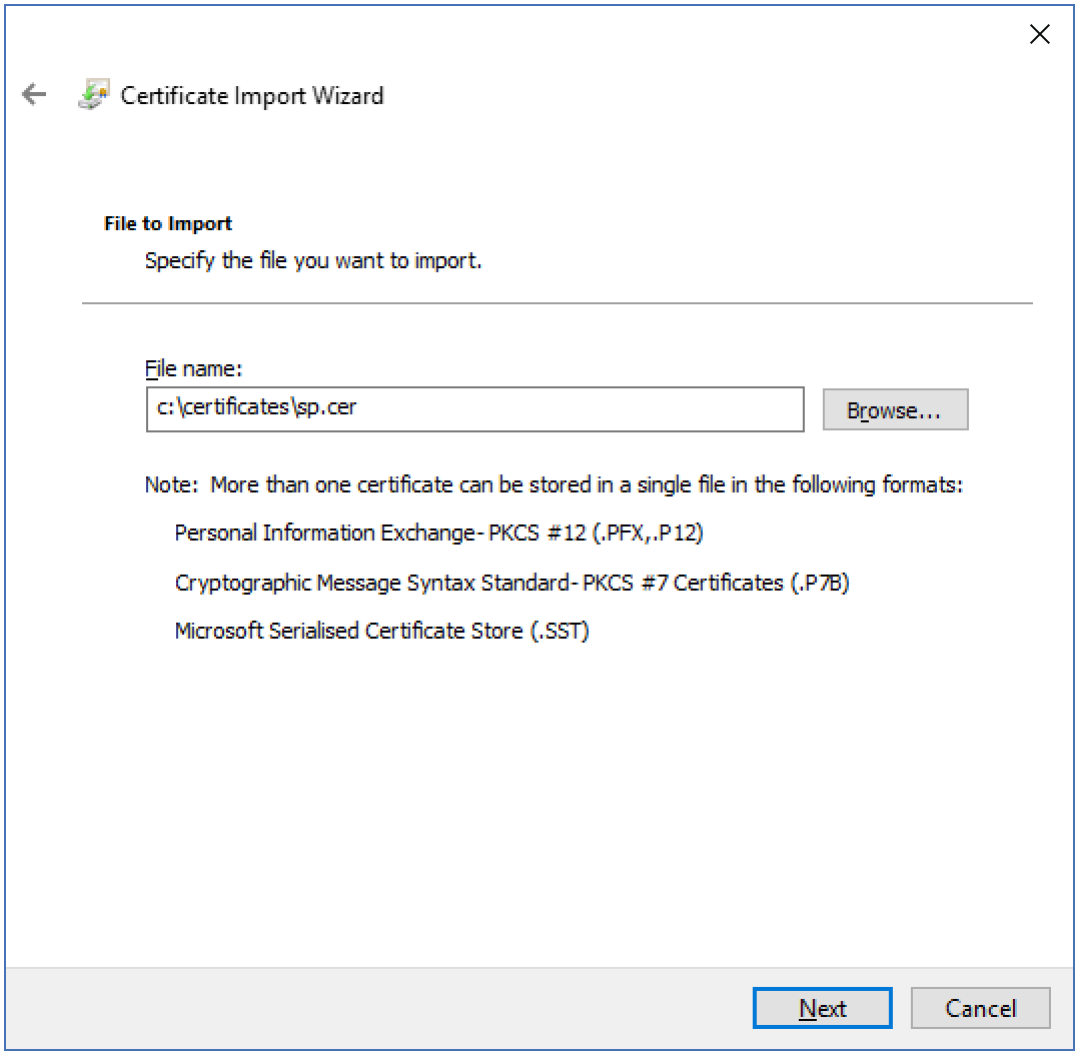


Importing Certificates

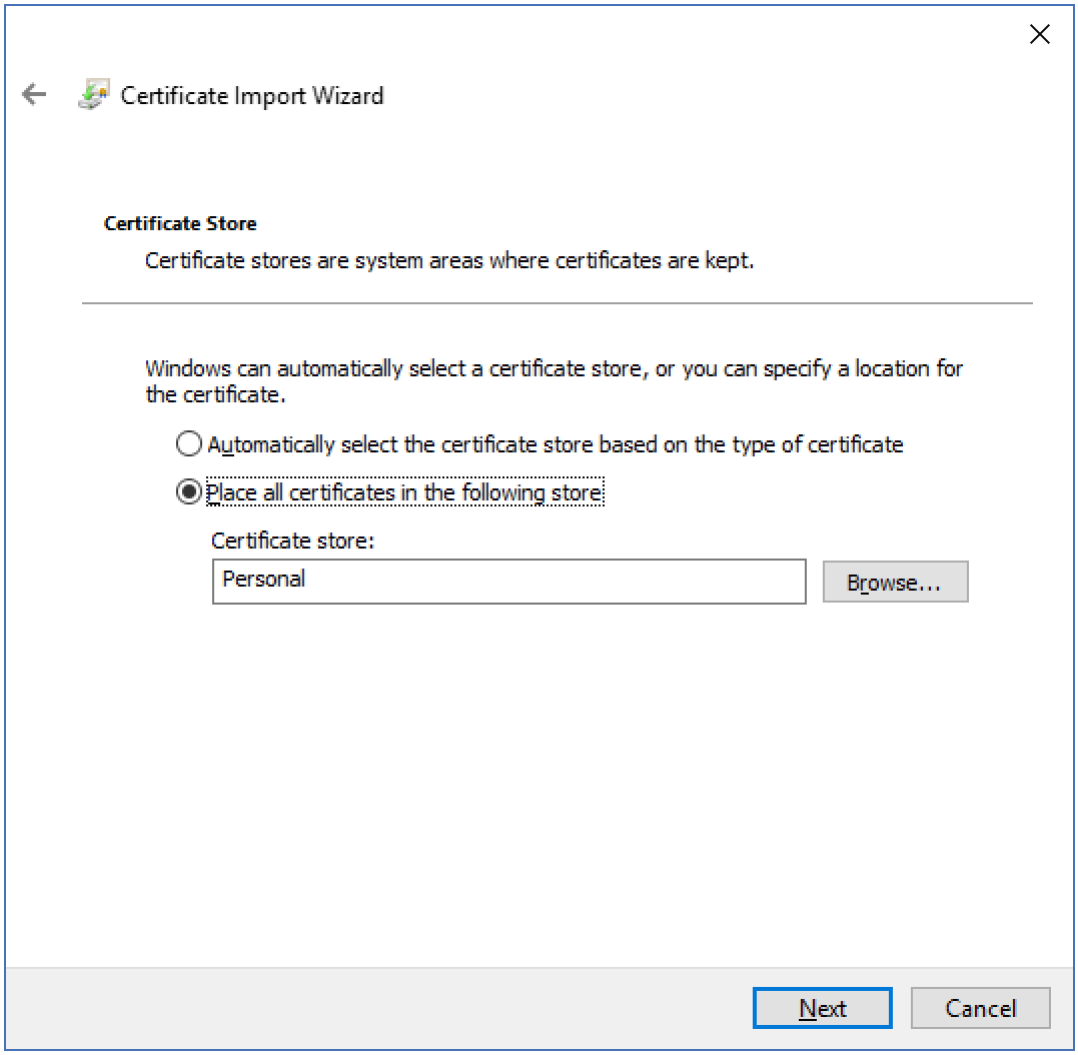
To import a .PFX or .CER file into the Windows certificate store, select the Certificates node in the tree view and, from the main menu, select Action > All Tasks > Import.



Select the file to import. Either a .CER or .PFX file may be imported.



Choose the personal store location.

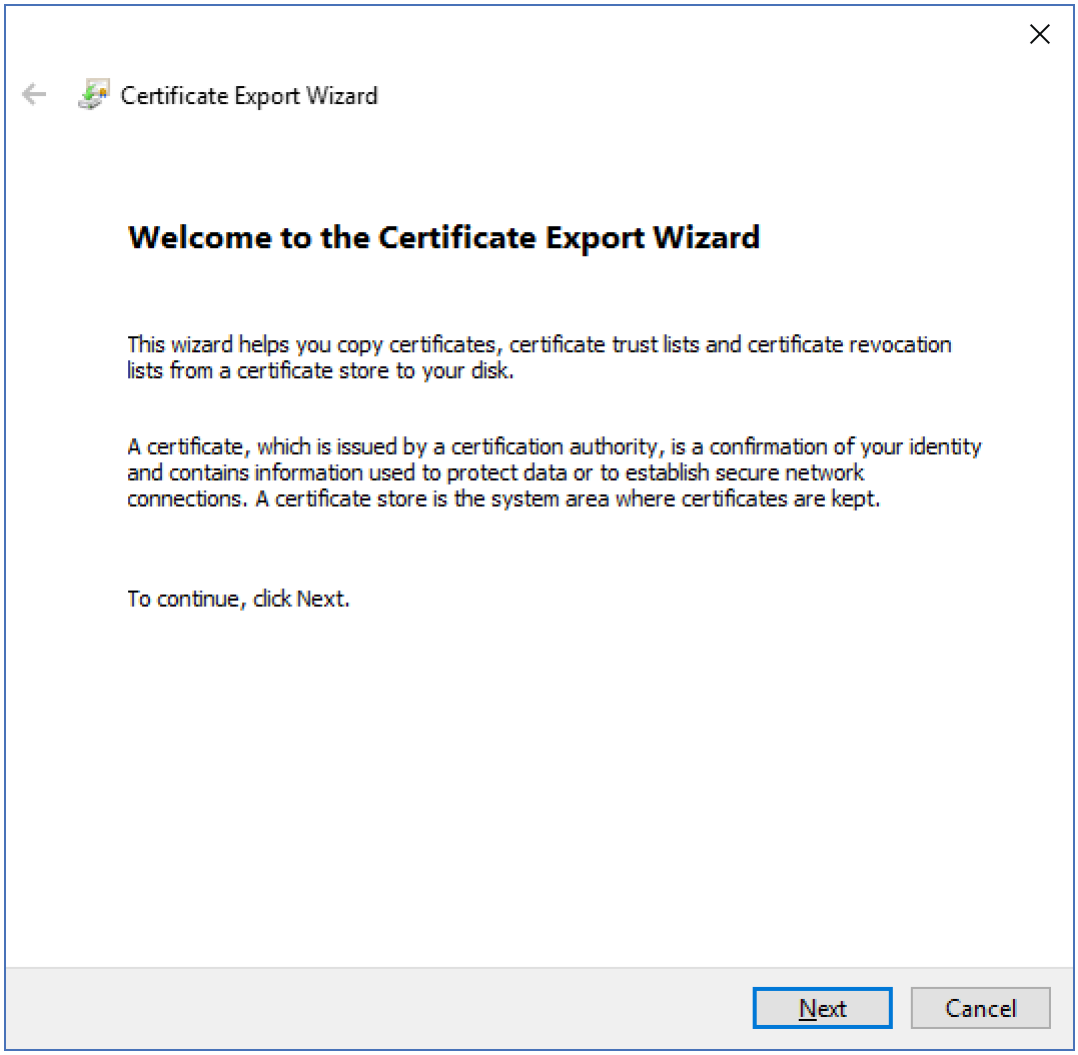


Click the Next and Finish buttons to complete the import.

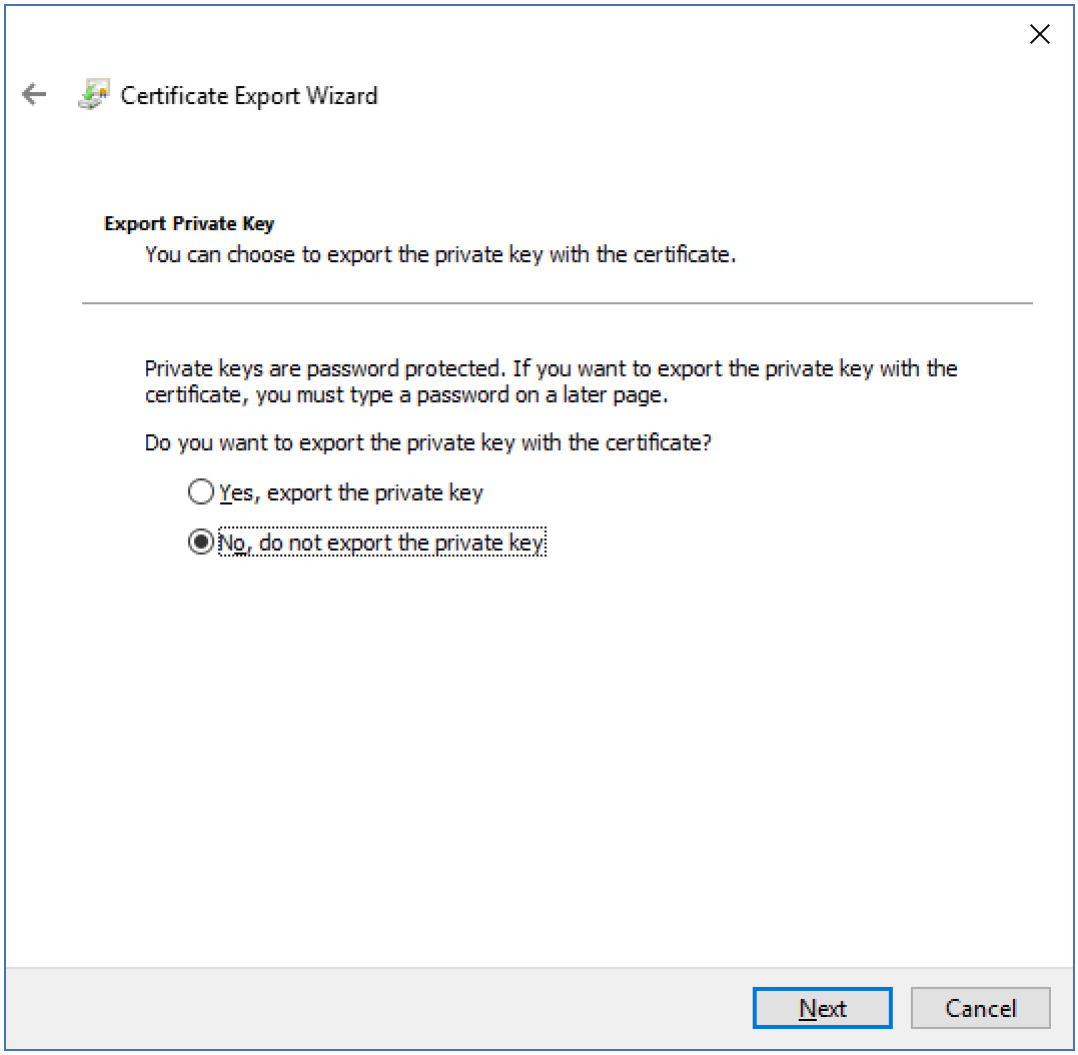
Note that for a .PFX file, the import wizard will prompt for its password.

Exporting Certificates

To export a certificate from the Windows certificate store to a .PFX or .CER file, select the certificate and, from the main menu, select Action > All Tasks > Export.

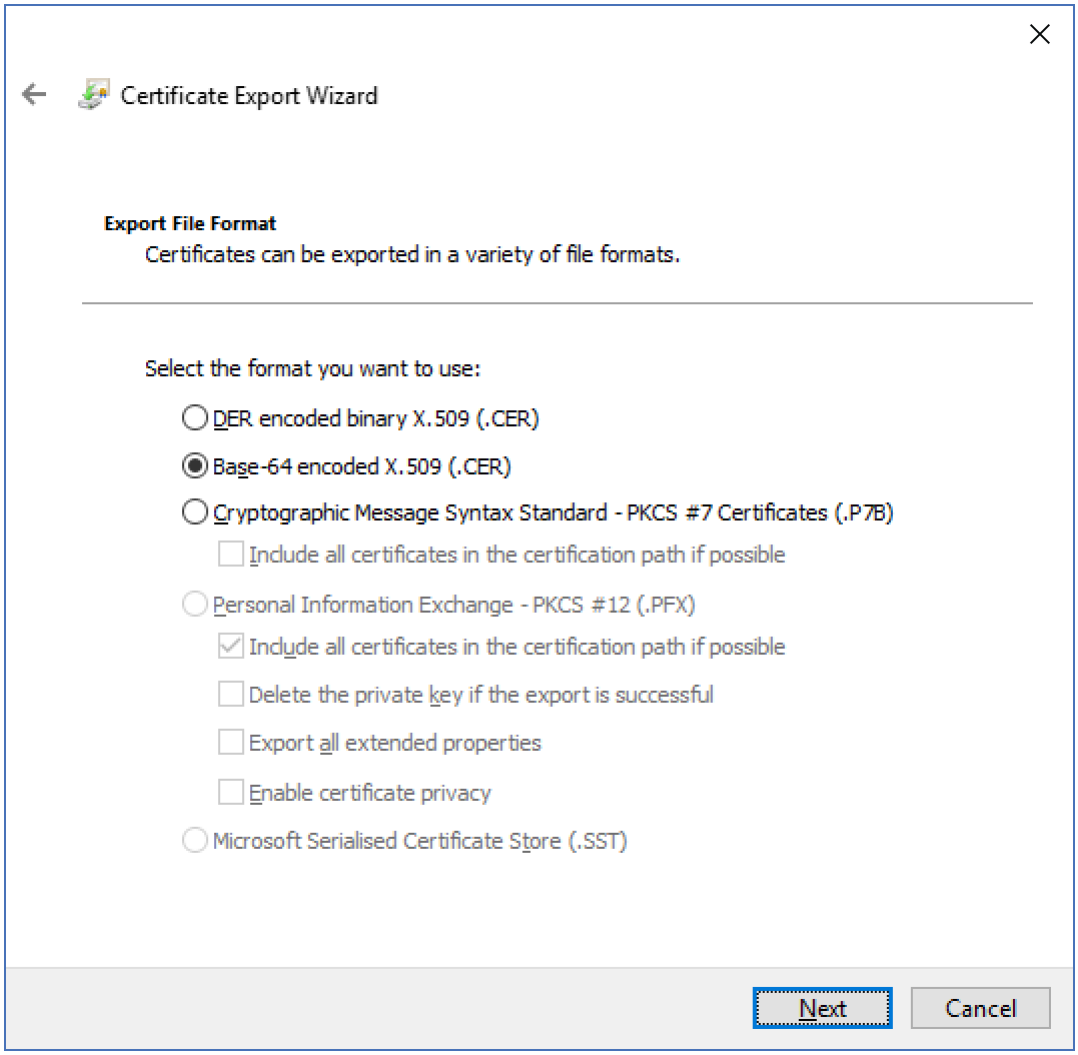


For a certificate with an associated private key, there's the option to export the private key as well.



To export a .PFX file, select the option to export the private key. To export a .CER file, select the option to not export the private key.

If the private key is not exported, the recommended format is base-64 encoded.



Click the Next button, specify the file location and click Finish to complete the export.

Note that for a .PFX file, the export wizard will prompt for a password.

Private Key Permissions

Private keys are protected by permissions. To use the private key, a process must have read permission.

To set permissions, select the certificate and, from the main menu, select Action > All Tasks > Manage Private Keys.

A dialog showing the current permissions is displayed.

For applications hosted in IIS, it's recommended that the IIS_IUSRS group be given read permission. If the application is running under an application pool whose account is not in this group, the permissions will have to be set explicitly for this account. The user or group to permit is dependent on the version of IIS and its configuration.

Generating Self-Signed Certificates

CreateSelfSignedCert

The CreateSelfSignedCert console application project may be used to create a self-signed certificate.

CreateSelfSignedCert may be run as follows.

```
CreateConfiguration.exe
```

It will prompt for various input required to create the certificate.

Subject distinguished name (eg CN=test):

The subject distinguished name (DN) is identifying information embedded in the certificate. The simplest DN is a common name (CN) whose format is “CN=<value>”.

A CN identifying your organization or application is recommended.

Key Size in bits [2048]:

Specify the key size.

Number of years before expiring [5]:

Specify the numbers of years before the certificate expires.

Certificate file name (eg test.cer):

Specify the certificate file name.

Private key file name (eg test.pfx):

Specify the private key file name.

Private key password:

Specify a password to protect the private key.

New-SelfSignedCertificate

Another option for generating self-signed certificates is to use PowerShell’s New-SelfSignedCertificate cmdlet.

https://technet.microsoft.com/en-us/itpro/powershell/windows/pkiclient/new-selfsignedcertificate

This must be run as an administrator to have permission to save certificates to the certificate store.

The subject name should reflect the name of your organization.

The expiry date may be set as required.

Note that to generate SHA-256, SHA-384 and SHA-512 XML signatures, the Microsoft Enhanced RSA and AES Cryptographic Provider must be specified.

For example:

```
New-SelfSignedCertificate
 -Subject "www.idp.com"
 -CertStoreLocation cert:\LocalMachine\My
 -Provider "Microsoft Enhanced RSA and AES Cryptographic Provider"
 -HashAlgorithm SHA256
 -KeyLength 2048
 -NotAfter 1/1/2050
```

Some or all certificate details may be retrieved using the Get-ChildItem cmdlet or equivalent synonym.

For example:

```
set-location Cert:\LocalMachine\My
get-childItem | select Subject,SerialNumber,Thumbprint
```

For example:

```
set-location Cert:\LocalMachine\My
get-childItem | select *
```

Export-Certificate

The Export-Certificate cmdlet may be used to export the certificate to a DER-encoded certificate file.

https://technet.microsoft.com/en-us/itpro/powershell/windows/pkiclient/export-certificate

For example:

```
Export-Certificate
 -Cert cert:\LocalMachine\My\295CB3430153889D6523554A002134425167F16E
 -FilePath c:\certs\idp.der
```

CertUtil

The exported certificate file is DER-encoded. The Microsoft utility, CertUtil, may be used to convert this to a base-64 encoded certificate file.

For example:

```
Certutil.exe -encode c:\certs\idp.der c:\certs\idp.cer
```

Export-PfxCertificate

The Export-PfxCertificate cmdlet may be used to export the certificate and private key to a PFX file.

https://technet.microsoft.com/en-us/itpro/powershell/windows/pkiclient/export-pfxcertificate

For example:

```
$password = ConvertTo-SecureString -String "password" -Force -AsPlainText

Export-PfxCertificate
 -Cert cert:\LocalMachine\My\295CB3430153889D6523554A002134425167F16E
 -FilePath c:\certs\idp.pfx
 -ChainOption EndEntityCertOnly
 -Password $password
```

Useful PowerShell Commands

Creating CER from PFX

For example:

```
Get-PfxCertificate -FilePath idp.pfx | Export-Certificate -FilePath idp.der
```

The exported certificate file is DER-encoded. The Microsoft utility, CertUtil, may be used to convert this to a base-64 encoded certificate file.

For example:

```
Certutil.exe -encode c:\certs\idp.der c:\certs\idp.cer
```

Useful OpenSSL Commands

The OpenSSL console application may be used to perform various certificate file commands.

https://www.openssl.org/

Windows binaries are available for download. Refer to the OpenSSL Wiki.

https://wiki.openssl.org/index.php/Binaries

The latest 64-bit Windows non-light installer at Shining Light Productions OpenSSL Installers is recommended.

https://slproweb.com/products/Win32OpenSSL.html

Creating Self-Signed PFX

For example:

```
openssl req -x509 -newkey rsa:4096 -days 730 -sha256 -subj "/CN=www.idp.com" -keyout
key.pem -out cert.pem
```

```
openssl pkcs12 -inkey key.pem -in cert.pem -export -out idp.pfx -CSP "Microsoft Enhanced RSA and AES Cryptographic Provider"
```

The days option specifies how long the certificate is valid.

If the subj[ect] option is omitted you are prompted for these details.

The Microsoft Enhanced RSA and AES Cryptographic Provider is required for SHA-2 support.

Creating CER from PFX

For example:

```
openssl pkcs12 -in idp.pfx -out idp.cer -nokeys -clcerts
```

The nokeys option excludes private keys from being output.

The clcerts option outputs client rather than CA certificates.

The generated PEM encoded file includes bag attributes. These may be removed if preferred.

```
openssl x509 -in idp.cer -out idp2.cer
```

Converting PEM to DER

For example:

```
openssl x509 -in idp-pem.cer -outform DER -out idp-der.cer
```

Converting DER to PEM

For example:

```
openssl x509 -inform der -in idp-der.cer -out idp-pem.cer
```