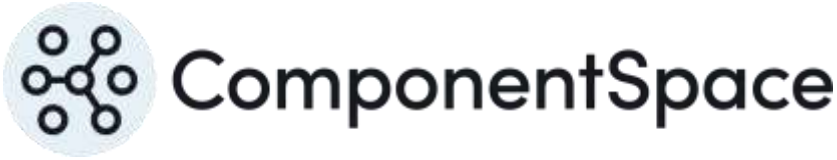

ComponentSpace

OpenID Connect for ASP.NET Core

Certificate Guide

www.componentspace.com

Contents

Introduction

X.509 certificates are used to secure messages and tokens sent between the OpenID Connect (OIDC) client and provider.

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

Transport Layer Security Certificates

The OpenID Connect specification recommends that all communications are over HTTPS.

As most use cases include some OpenID Connect messages being exchanged between the OpenID client and 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 (e.g. self-signed certificates) will result in the browser presenting a warning message to the user.

Signature and Encryption Certificates

Certificates used for signature and/or 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 few trusted parties.

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

HTTPS Shared Certificates

CA-issued certificates for HTTPS endpoints also may be used for signature and/or 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

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.

An OpenID provider certificate stored on the file system may be specified in the provider configuration.

```
"ProviderCertificates": [
 {
  "FileName": "certificates/op.pfx",
  "Password": "password"
 }
]
```

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

A client certificate stored on the file system may be specified in the client configuration.

```
"ClientCertificates": [
 {
  "FileName": "certificates/client.cer"
 }
```

```
]
```

A client 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 provider certificate stored in the Windows certificate store may be specified in the provider 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.

```
"ProviderCertificates": [
  {
    "SerialNumber": "331df5506e114644"
  }
]
```

Alternatively, the certificate may be identified by its thumbprint.

```
"ProviderCertificates": [
  {
    "Thumbprint": "2ea91613e2eb2a060a7d0b35975eb436122c753e"
  }
]
```

Or the certificate may be identified by its subject name.

```
"ProviderCertificates": [
  {
    "SubjectName": "ExampleOpenIDProvider"
  }
]
```

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

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

```
"ClientCertificates": [
  {
    "Thumbprint": "20e25105f62bef6c69a9b6c5f2ad5d6633efb243"
  }
```

```
]
```

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.

```
"ProviderCertificates": [
 {
  "StoreLocation: "CurrentUser"
  "Thumbprint": "2ea91613e2eb2a060a7d0b35975eb436122c753e"
 }
]
```

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

```
"ProviderCertificates": [
 {
  "StoreName: "WebHosting"
  "Thumbprint": "2ea91613e2eb2a060a7d0b35975eb436122c753e"
 }
]
```

Certificate Strings

Certificates may be specified as base-64 encoded strings.

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

A provider certificate string may be specified in the configuration.

```
"ProviderCertificates": [
 {
  "String": "MIIC/jCCAeagAwIBAgIQ…",
  "Password": "password"
 }
]
```

A 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:

```
$bytes = [System.IO.File]::ReadAllBytes("op.pfx")
[System.Convert]::ToBase64String($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\op.pfx c:\certs\b64-op.pfx
```

A client certificate string may be specified in the configuration.

```
"ClientCertificates": [
  {
    "String": "MIIJegIBAzCCCTYGC…"
  }
]
```

A client 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\client.cer c:\certs\b64-client.cer
```

Application Configuration

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

These certificates are accessed through Microsoft's IConfiguration interface 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.

```
"ProviderCertificates": [
  {
    "Key": "OP"
    "Password": "password"
  }
]
```

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 name specified in the key vault is used as the certificate configuration key.

A password is not required.

```
"ProviderCertificates": [
  {
    "Key": "ExampleOpenIDProvider"
  }
]
```

Key Vault Configuration Provider

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

https://docs.microsoft.com/en-us/aspnet/core/security/key-vault-configuration?tabs=aspnetcore2x

Certificate Status

By default, any certificate specified is assumed to be available for use.

However, it is possible to identify certificates as being active, retired or for future use. This assists with certificate rollover.

Only active certificates are used for signature generation or encryption.

Retired certificates are previously active certificates that are no longer in use.

Future certificates will become active at some future point in time.

For example, the first certificate has been retired. The second certificate is active. The third certificate will become active at some point.

All certificates are included when returning the provider's keys as part of discovery.

```
"ProviderCertificates": [
```

```
 {
  "Status": "Retired"
  "SubjectName": "january-ExampleOpenIDProvider"
 },
 {
  "Status": "Active"
  "SubjectName": "may-ExampleOpenIDProvider"
 },
 {
  "Status": "Future"
  "SubjectName": "september-ExampleOpenIDProvider"
 }
]
```

Certificate Use

By default, any certificate specified is assumed to be available for signature support and encryption support.

However, it is possible to limit the use of a certificate to only signature support or encryption support.

For example, the first certificate is used for signature verification. The second certificate is used for encryption.

```
"ClientCertificates": [
 {
  "Use": "Signature"
  "SubjectName": "signature-ExampleOpenIDClient"
 },
 {
  "Use": "Encryption"
  "SubjectName": "encryption-ExampleOpenIDClient "
 }
]
```

Certificate Rollover

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

Through OpenID Connect's discovery mechanism, clients may retrieve the OpenID provider's JSON Web Key Set [JWK] document.

This ensures clients can retrieve any new public keys associated with a certificate rollover.

Certificate Validation

By default, certificates are checked to see if they've expired. Additional checks may be enabled through the CertificateValidationOptions. Refer to the Developer Guide for more information.

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

Therefore, consideration must be given to the performance impact associated with certificate validation during SSO.

If further and potentially more expensive validation is 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 potential coordination with users of the certificate.

Generating Self-Signed Certificates

New-SelfSignedCertificate

Self-signed certificates may be generated using 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 signatures, the Microsoft Enhanced RSA and AES Cryptographic Provider must be specified.

For example:

```
New-SelfSignedCertificate
  -Subject "ExampleOpenIDProvider"
  -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\op.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\op.der c:\certs\op.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\op.pfx
 -ChainOption EndEntityCertOnly
 -Password $password
```

Certificate File Formats

Windows and the .NET framework support several 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.