# Introduction

This project provides guided examples of how to setup Azure based projects to sign code within an Azure CI pipeline using a code signing certificate stored in key vault.

# Pre-Requisites

* An active Azure subscription where you can manage Azure Active Directory
* Access to an Azure DevOps environment where you have permissions to run build pipelines and leverage service connections.
* PowerShell core 7 or greater installed locally.

# Common setup tasks

Clone the repo located [here](tbd) and upload it into your own Azure DevOps environment. Please use the same repo name as found in the demo to maintain consistency with the steps to follow.

Get and save {yourSubscriptionId} and {yourSubscriptionName} for the subscription that will be hosting the certificate key vault.

Get and save {yourTenantId} and {yourTenantName} from the properties of your Azure Active Directory Tenant.

[Create a resource group](https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-portal#create-resource-groups) within you Azure subscription named {yourResourceGroupName}.

[Create an app registration](https://docs.microsoft.com/en-us/azure/active-directory/develop/howto-create-service-principal-portal) within you Azure AD with the following configuration. This will be used as the identity for your Azure DevOps service connection when accessing the key vault.

1. Name: {yourAppRegName}.
2. Supported account types: Select “Accounts in this organizational directory only”

Save your {yourAppRegName} application Id, {yourAppRegAppId}, for later configuration.

[Add a secret](https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app#add-credentials) to your app registration. Ensure you copy the provided {secretString} prior to leaving the page because this is the only time that the secret is visible to the user. This is needed to setup your service connection.

[Add a “Contributor” role assignment](https://docs.microsoft.com/en-us/azure/role-based-access-control/role-assignments-portal#add-a-role-assignment) to {yourResourceGroupName} resource group for {yourAppRegName} security principal.

[Create a key vault](https://docs.microsoft.com/en-us/azure/key-vault/general/quick-create-portal#create-a-vault) with the following configuration within the {yourResourceGroupName} resource group

1. Basics tab
   1. Subscription: Choose your subscription containing {yourResourceGroupName} if it isn’t already defaulted.
   2. Resource Group: {yourResourceGroupName}
   3. Name: {yourKeyVaultName}. This value must be unique across all key vaults so you will have to choose the name to ensure uniqueness.
2. Access policy tab
   1. Select “Azure Resource Manager for template deployment” under “Enable Access to:”
   2. Click “+ Add Access Policy”
      1. Select “key permissions”
         1. Get
         2. Sign
      2. Select “secret permissions”
         1. Get
      3. Select “certificate permissions”
         1. Get
      4. Service Principal
         1. Select {yourAppRegName}
         2. Click “Select”
      5. Click “add”

Create a self-signed code signing certificate.

1. Open a PowerShell core window with elevated permissions
2. Run scripts\New-SelfSignedCert.ps1. This script will create a self-signed code signing certificate pfx file in the local user’s Documents directory with the name “OltivaBikesCodeSigning.pfx”. This certificate file contains no password protection. This certificate will also be found in the user’s personal certificate store. The certificate in the certificate store can be deleted.

[Upload the code signing certificate](https://docs.microsoft.com/en-us/azure/key-vault/certificates/tutorial-import-certificate#import-a-certificate-to-key-vault) into key vault with the following configuration:

1. Certificate Name: {yourCertificateName}
2. Upload Certificate File: “OltivaBikesCodeSigning.pfx”

Copy the displayed {certificateThumbprint} for use later when creating the Azure DevOps variable groups.

[Create a variable group](https://docs.microsoft.com/en-us/azure/devops/pipelines/library/variable-groups?view=azure-devops&tabs=yaml) in your Azure DevOps pipeline library named “Deployment Environment” with the following variables:

1. ServiceConnectionClientId: contains {yourAppRegAppId}
2. ServiceConnectionClientSecret: contains {secretString}. Enable variable type secret by clicking the Lock Icon far right on the entry
3. ServiceConnectionName: contains {yourAppRegName}
4. TenantId: contains {yourTenantId}
5. TenantName: contains {yourTenantName}.onmicrosoft.com

[Create a variable group](https://docs.microsoft.com/en-us/azure/devops/pipelines/library/variable-groups?view=azure-devops&tabs=yaml) in your Azure DevOps pipeline library named “Code Signing” with the following variables:

1. CertKeyVaultName: containing {yourKeyVaultName}
2. CertKeyVaultUrl: containing https://{yourKeyVaultName}.vault.azure.net/
3. CertName: containing {yourCertificateName}
4. CertThumbprint: containing {certificateThumbprint}
5. Publisher: containing “CN=Oltiva Bikes Inc., O=Oltiva Bikes Incorporated, C=US” from the code signing certificate creation script
6. PublisherName: containing “Oltiva Bikes Inc.”, the CN value from the code signing certificate creation script.
7. TimestampUrl: containing the timestamp url you use for code signing. You can use DigiCert’s timestamp service at <http://timestamp.digicert.com>.

[Create a service connection](https://docs.microsoft.com/en-us/azure/devops/pipelines/library/service-endpoints?view=azure-devops&tabs=yaml) for Azure DevOps

1. Open the Azure DevOps team project to where you uploaded the cloned demo project code.
2. Select the Project Settings icon (“gear” at the bottom left).
3. Select “Service Connections” in the left navigation panel.
4. Click “New service connection”
5. Select the “Azure Resource Manager” connection type and click “Next”
6. Select “Service principal (manual)” and click “Next”
7. Ensure “Environment” is set to “Azure Cloud”
8. Ensure “Scope Level” is set to “Subscription”
9. In “Subscription Id”, enter {yourSubscriptionId}
10. In “Subscription Name”, enter {yourSubscriptionName}
11. In “Service Principal Id”, enter {yourAppRegAppId}
12. Select “Service principal key” for Credential
13. In “Service principal key”, enter {secretString}
14. In “Tenant Id”, enter {yourTenantId}
15. In “Service connection name”, enter {yourAppRegName}
16. Click “Verify and save”

# Signing a Console Application

Signing console applications within a ci pipeline uses a tool called the AzureSignTool whose source code can be found [here](https://github.com/vcsjones/AzureSignTool/blob/v2.0.17/README.md). This tool is configured and works very similar to the console app SignTool provided by Microsoft.

[Create an Azure pipeline](https://docs.microsoft.com/en-us/azure/devops/pipelines/create-first-pipeline?view=azure-devops&tabs=net%2Ctfs-2018-2%2Cbrowser)

1. Log in to your Azure DevOps organization and navigate to your project
2. Go to pipelines, and then select “New Pipeline”
3. Select “Use the classic editor” at the bottom of the repo list.
4. Select the type of repo to which you uploaded your demo clone. We are assuming “Azure Repos Git” for the remainder of this sample
5. Select the Team project to where you uploaded your demo clone.
6. Select the Repository (ex. “azure-code-signing-cicd-demo”)
7. Select the branch (ex. “main” or “master”)
8. Click “Continue”
9. Select the “.NET Desktop” template
10. Click “Apply”

Add variables

1. In “Pipeline variables” add/update:

|  |  |
| --- | --- |
| Name | Value |
| AppPackageName | ConsoleApp |
| AppToSign | ConsoleApp.exe |
| BuildConfiguration | release |
| BuildPlatform | any cpu |
| TargetProject | src\ConsoleApp |

1. In “Variable groups”, link variable group:
   1. “Deployment environment”
   2. “Code Signing”

Configure the Azure pipeline

1. At the “Pipeline” level
   1. Update pipeline name to something more descriptive, like “BuildAndSignConsoleApp”
   2. Update the Agent specification to use “windows-2019”
   3. Update the Solution to “auzre-code-signing-cicd-demo.sln”
2. In the “Use NuGet” task
   1. Update “Version of NuGet.exe to install” to “5.7.0”
3. In the “Build project…” task
   1. Update the “MSBuild Arguments” to:

/t:$(TargetProject):Rebuild

1. Add a new “Command line” task named “Install AzureSignTool” after VsTest task
   1. Paste into the script box

set DOTNET\_SKIP\_FIRST\_TIME\_EXPERIENCE=true

dotnet tool install --global AzureSignTool --version 2.0.17

1. Add a new “Command line” task named “Sign Application” after the task created above
   1. Paste into the script box

AzureSignTool.exe sign ^

-kvu $(CertKeyVaultUrl) ^

-kvi $(ServiceConnectionClientId) ^

-kvs $(ServiceConnectionClientSecret) ^

-kvc $(CertName) ^

-tr $(TimestampUrl) ^

-v ^

$(System.DefaultWorkingDirectory)\$(TargetProject)\bin\$(BuildConfiguration)\$(AppToSign)

1. In the “Copy Files to:…” task
   1. Update the “Source Folder” to “$(System.DefaultWorkingDirectory)\$(TargetProject)\bin\$(BuildConfiguration)\”
2. In the “Publish Artifact:…” task
   1. Update “Artifact name” to “$(AppPackageName)”

# Signing A ClickOnce Application

Signing a ClickOnce application in a ci pipeline requires pulling the certificate from the Azure key vault and saving it into local pfx file; and overriding a standard build task within MSBuild, “SignFile”. The override can be found in the demo repo within the scripts directory in a file called FileSign.targets. This override task evaluates of a “CertificateFile” parameter is provided in the MSBuild Arguments list. If ‘yes’, then the alternate signing mechanism is invoked; otherwise, it uses the standard coding for signing. This override is implemented by altering the csproj file for the target application, adding an Import node at the bottom:

…

<Import Project="$(SolutionDir)\scripts\FileSign.targets" />

</Project>

[Create an Azure pipeline](https://docs.microsoft.com/en-us/azure/devops/pipelines/create-first-pipeline?view=azure-devops&tabs=net%2Ctfs-2018-2%2Cbrowser)

1. Log in to your Azure DevOps organization and navigate to your project
2. Go to pipelines, and then select “New Pipeline”
3. Select “Use the classic editor” at the bottom of the repo list.
4. Select the type of repo to which you uploaded your demo clone. We are assuming “Azure Repos Git” for the remainder of this sample
5. Select the Team project to where you uploaded your demo clone.
6. Select the Repository (ex. “azure-code-signing-cicd-demo”)
7. Select the branch (ex. “main” or “master”)
8. Click “Continue”
9. Select the “.NET Desktop” template
10. Click “Apply”

Add variables

1. In “Pipeline variables” add/update:

|  |  |
| --- | --- |
| Name | Value |
| AppPackageName | ClickOnceApp |
| BuildConfiguration | release |
| BuildPlatform | any cpu |
| PfxFilename | $(System.DefaultWorkingDirectory)\CertFile.pfx |
| PublishPath | $(System.DefaultWorkingDirectory)\$(TargetProject)\bin\$(BuildConfiguration)\app.publish |
| TargetProject | src\ClickOnceApp |

1. In “Variable groups”, link variable group:
   1. “Deployment environment”
   2. “Code Signing”

Configure the Azure pipeline

1. At the “Pipeline” level
   1. Update pipeline name to something more descriptive, like “BuildAndSignClickOnceApp”
   2. Update the Agent specification to use “windows-2019”
   3. Update the Solution to “auzre-code-signing-cicd-demo.sln”
2. In the “Use NuGet” task
   1. Update “Version of NuGet.exe to install” to “5.7.0”
3. Add a new “Azure PowerShell” task named “Get Code Sign Cert” after “NuGet restore”
   1. In “Azure Subscription” enter “$(ServiceConnectionName)”
   2. Select “Inline Script” for “Script Type”
   3. Paste into script box

$KeyVaultName = "$(CertKeyVaultName)"

$CertName = "$(CertName)"

$CertFilename = "$(PfxFilename)"

# get cert secret from key vault

$secret = Get-AzKeyVaultSecret -VaultName $KeyVaultName -Name $cert.Name

$ssPtr = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($secret.SecretValue)

$secretValueText = '';

try {

$secretValueText = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR($ssPtr)

} finally {

[System.Runtime.InteropServices.Marshal]::ZeroFreeBSTR($ssPtr)

}

$secretByte = [Convert]::FromBase64String($secretValueText)

# create pfx file structure with cert secret information

$x509Cert = [System.Security.Cryptography.X509Certificates.X509Certificate2]::new($secretByte, "", "Exportable,PersistKeySet")

$type = [System.Security.Cryptography.X509Certificates.X509ContentType]::Pfx

$pfxFileByte = $x509Cert.Export($type, "")

# Write to a file

[System.IO.File]::WriteAllBytes($CertFilename, $pfxFileByte)

1. In the “Build project…” task
   1. Update the “MSBuild Arguments” to:

/t:$(TargetProject):Publish /p:SignManifest=true /p:SignAssembly=true /p:PublisherName="$(PublisherName)" /p:PublishUrl="$(PublishPath)\\" /p:ManifestKeyFile="$(PfxFilename)" /p:CertificateFilename="$(PfxFilename)" /p:CertificatePassword="" /p:ManifestCertificateThumbprint="$(CertThumbprint)" /p:TimestampUrl="$(TimestampUrl)"

1. In the “Copy Files to:…” task
   1. Update the “Source Folder” to “$(PublishPath)\”
2. In the “Publish Artifact:…” task
   1. Update “Artifact name” to “$(AppPackageName)”

# Signing A UWP Application

Signing a UWP application in a ci pipeline is very much like signing a ClickOnce application. The process requires downloading the cert file from the key vault, overriding the SignFile MSBuild task, and this time updating the Package.appxmanifest file with the Publisher that matches that in the code signing certificate. The Package.appxmanifest file is updated leveraging xml document query. The override is implemented by altering the csproj file for the target application, adding an Import node at the bottom:

…

<Import Project="$(SolutionDir)\scripts\FileSign.targets" />

</Project>

[Create an Azure pipeline](https://docs.microsoft.com/en-us/azure/devops/pipelines/create-first-pipeline?view=azure-devops&tabs=net%2Ctfs-2018-2%2Cbrowser)

1. Log in to your Azure DevOps organization and navigate to your project
2. Go to pipelines, and then select “New Pipeline”
3. Select “Use the classic editor” at the bottom of the repo list.
4. Select the type of repo to which you uploaded your demo clone. We are assuming “Azure Repos Git” for the remainder of this sample
5. Select the Team project to where you uploaded your demo clone.
6. Select the Repository (ex. “azure-code-signing-cicd-demo”)
7. Select the branch (ex. “main” or “master”)
8. Click “Continue”
9. Select the “.NET Desktop” template
10. Click “Apply”

Add variables

1. In “Pipeline variables” add/update:

|  |  |
| --- | --- |
| Name | Value |
| AppPackageName | $(ProjectName) |
| AppxBundlePlatforms | x86|x64|ARM |
| AppxPackagePath | $(System.DefaultWorkingDirectory)\$(TargetProject)\AppPackages\\ |
| BuildConfiguration | release |
| BuildPlatform | x86 |
| PfxFilename | $(System.DefaultWorkingDirectory)\CertFile.pfx |
| ProjectName | UWPApp |
| TargetProject | src\$(ProjectName) |

1. In “Variable groups”, link variable group:
   1. “Deployment environment”
   2. “Code Signing”

Configure the Azure pipeline

1. At the “Pipeline” level
   1. Update pipeline name to something more descriptive, like “BuildAndSignUWPApp”
   2. Update the Agent specification to use “windows-2019”
   3. Update the Solution to “auzre-code-signing-cicd-demo.sln”
2. Add a new “PowerShell” task named “Update Package.appxmanifest with signing publisher” as the first task
   1. Select “Inline” type
   2. Paste into the script box

$xmlFile = "$(System.DefaultWorkingDirectory)\$(TargetProject)\Package.appxmanifest"

[xml]$xmlDoc = Get-Content $xmlFile -Raw

$xmlDoc.Package.Identity.Publisher = "$(Publisher)"

$xmlDoc.Save($xmlFile)

Write-Host "Updated '$($xmlFile)'."

1. In the “Use NuGet” task
   1. Update “Version of NuGet.exe to install” to “5.7.0”
2. Add a new “Azure PowerShell” task named “Get Code Sign Cert” after “NuGet restore”
   1. In “Azure Subscription” enter “$(ServiceConnectionName)”
   2. Select “Inline Script” for “Script Type”
   3. Paste into script box

$KeyVaultName = "$(CertKeyVaultName)"

$CertName = "$(CertName)"

$CertFilename = "$(PfxFilename)"

# get cert secret from key vault

$secret = Get-AzKeyVaultSecret -VaultName $KeyVaultName -Name $cert.Name

$ssPtr = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($secret.SecretValue)

$secretValueText = '';

try {

$secretValueText = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR($ssPtr)

} finally {

[System.Runtime.InteropServices.Marshal]::ZeroFreeBSTR($ssPtr)

}

$secretByte = [Convert]::FromBase64String($secretValueText)

# create pfx file structure with cert secret information

$x509Cert = [System.Security.Cryptography.X509Certificates.X509Certificate2]::new($secretByte, "", "Exportable,PersistKeySet")

$type = [System.Security.Cryptography.X509Certificates.X509ContentType]::Pfx

$pfxFileByte = $x509Cert.Export($type, "")

# Write to a file

[System.IO.File]::WriteAllBytes($CertFilename, $pfxFileByte)

1. In the “Build project…” task
   1. Update the “MSBuild Arguments” to:

/t:$(TargetProject):Rebuild /p:AppxBundlePlatforms="$(appxBundlePlatforms)" /p:AppxBundle=Always /p:UapAppxPackageBuildMode=SideloadOnly /p:AppxPackageSigningEnabled=true /p:PackageCertificateKeyFile="$(PfxFilename)" /p:CertificateFile="$(PfxFilename)" /p:CertificatePassword="" /p:PackageCertificateThumbprint="$(CertThumbprint)" /p:AppxPackageSigningTimestampServerUrl="$(TimestampUrl)"

1. In the “Copy Files to:…” task
   1. Update the “Source Folder” to “$(AppxPackagePath)”
2. In the “Publish Artifact:…” task
   1. Update “Artifact name” to “$(AppPackageName)”

# References:

1. https://blog.danskingdom.com/creating-a-pfx-certificate-and-applying-it-on-the-build-server-at-build-time/
2. https://blog.danskingdom.com/continuously-deploy-your-clickonce-application-from-your-build-server/
3. https://github.com/erikest/SignClickOnce
4. https://stackoverflow.com/questions/25004056/resign-clickonce-manifest-using-mage-exe
5. https://stackoverflow.com/questions/53291017/securely-signing-clickonce-applications-in-azure-devops-pipeline