



UWP



Digital signatures



WINDOWS > GUIDES > SIGNATURES > DIGITAL SIGNATURES

Add digital signatures to PDFs in Windows

Nutrient enables signing both existing signature form elements and documents without a signature form element. If there is no signature form field in the current document, Nutrient gives you the byte range and a hash representation of the current state of the document, so you can digitally sign it by obtaining a DER [PKCS#7 container](#) from either the byte range or the hash and then applying that DER PKCS#7 container to a new, ad hoc invisible signature form field.

You're only responsible for providing the signing service that will receive the byte range and the hash as input values and return the DER PKCS#7 container to be applied to the prepared signature form field. Our implementation allows you to produce, validate, and display digitally signed documents in a totally flexible way.



If you want to use the Digital Signatures component, make sure it's included in your license. Contact Sales for more information.

Approval and certification signatures

PDF documents mainly support two types of digital signatures: approval signatures, and certification signatures. Approval signatures are used to indicate that a signer agrees with or acknowledges the contents of a document. A single document can contain multiple approval signatures. Meanwhile, certification signatures restrict the kind of changes that can be applied to a document once it's signed. A PDF document only allows one certification signature. Nutrient provides support for approval signatures. For certification signatures, contact [Support](#).



ASK AI

Creating a digital signature

Adding a digital signature on a PDF document is both reliable proof of the document's origin and protection against modification by third parties.

To create a digital signature, you need two things.

- ⌘ First, you need an **X.509** certificate that contains your **public key** and your **signer information**. Nutrient supports PEM-encoded X.509 certificates, as well as PEM-encoded PKCS#7 certificates. You can verify if a PKCS#7 certificate file is correctly PEM-encoded by using the OpenSSL command-line tool as follows:

```
openssl pkcs7 -noout -text -print_certs -in example.p7b
```



The above command will print an error message if “example.p7b” is not a PEM-encoded PKCS#7 certificate or certificate chain.

To verify if a PKCS#7 certificate file is correctly DER encoded, you can use this command instead:

```
openssl pkcs7 -inform der -noout -text -print_certs -in example.p7b
```



The above command will print an error message if “example.p7b” is not a DER-encoded PKCS#7 certificate or certificate chain.

- ⌘ Second, you need your **private key**. A self-signed private key and certificate pair can be created with the command shown in the previous section.

Single-step or two-step signing

With Nutrient UWP SDK, there are two strategies for digitally signing documents. You can provide the certificate and private key in PEM formats and sign the document in a single step. Alternatively, there's a two-step process where you can register an event handler that gets called by the framework to retrieve the signature. In that handler, you can generate the signature yourself and provide that to Nutrient UWP SDK. The two-step process can be useful for workflows where you may wish to perform the signing with specialized hardware devices or sign remotely, etc. It also has the benefit of ensuring that Nutrient doesn't need access to the private key that ultimately will be used to produce the signature value, leaving you complete freedom to choose which strategy to use to manage its lifecycle and security.

How it works

Under the hood, the process of signing a document is divided into three phases:

- 1 Nutrient prepares the document for a signature, adding an invisible form field that will contain the signature value.
- 2 Nutrient then either generates the signature in a PKCS#7 container internally or fires the two-step signature signing event handler for you to generate the signature.
- 3 Nutrient applies the signature and saves the document.

If the signing process is successful, the document is reloaded with the new invisible digital signature added to it.

During the call to `Document.SignAndSaveAsync`, and until either the DER PKCS#7 container is generated and Nutrient applies the signature to the document or the process is disregarded due to a rejection from the callback, all UI interactions from the user are disabled. This ensures that no modifications are made to the document while a new digital signature is about to be added to it.

Single-step signing example

```
1 // Nutrient UWP SDK supports certificates and private keys in PEM format.
2 var certificate = await GetTestAssetTextAsync("certificate.pem");
3 var privateKey = await GetTestAssetTextAsync("private-key.pem");
4
5 var doc = await Document.OpenDocumentAsync(source);
6
7 // Sign the document. SHA256 or greater is recommended.
8 await doc.SignAndSaveAsync(HashAlgorithm.Sha256, certificate, privateKey, new {
9     {
10         SignerName = "James Swift",
11         SignatureReason = "Example Signature",
12         SignatureLocation = "Vienna, Austria"
13     }
14 });
```

Two-step signing example

```
1 // Nutrient UWP SDK supports certificates and private keys in PEM format.
2 var certificate = await GetTestAssetTextAsync("certificate.pem");
3 var privateKey = await GetTestAssetTextAsync("private-key.pem");
```

```

4
5 var doc = await Document.OpenDocumentAsync(source);
6
7 // Register the two-step signature signer event handler.
8 doc.TwoStepSignatureSigning += async (deferral, signingData) =>
9 {
10     var certificate = GetCertificate();
11     var privateKey = GetPrivateKey();
12
13     // Hash the PDF data. SHA256 or greater is recommended.
14     byte[] hash;
15     using (var sha256 = new SHA256CryptoServiceProvider())
16     {
17         hash = sha256.ComputeHash(signingData.FileContents.AsStreamForRead());
18     }
19
20     // You can compare your own hash with a hash generated by Nutrient from the
21     if (CryptographicBuffer.EncodeToHexString(hash.AsBuffer()).ToUpperInvariant() != signingData.Hash)
22     {
23         throw new Exception("Hashes don't match.");
24     }
25
26     try
27     {
28         // Sign the hash.
29         // For this example, we use a utility method provided by Nutrient, but
30         // if you need to ensure that Nutrient doesn't have access to the private key,
31         var signature = await Utilities.GeneratePKCS7ContainerAsync(
32             hash.AsBuffer(),
33             certificate,
34             privateKey,
35             HashAlgorithm.Sha256);
36
37         signingData.Pkcs7Container = signature;
38     }
39     finally
40     {
41         // It is essential to complete the deferral.
42         // Read more about `Deferral`s in this blog post:
43         // /blog/async-callbacks-with-deferral-csharp/
44         deferral.Complete();
45     }
46 };
47
48 // Sign the document. SHA256 or greater is recommended.
49 // No need to provide a certificate or private key.
50 await doc.SignAndSaveAsync(HashAlgorithm.Sha256, "", "", new SignatureMetadata
51 {
52     SignerName = "James Swift",
53     SignatureReason = "Example Signature",
54     SignatureLocation = "Vienna, Austria"
55 });

```



Was this helpful?

 YES

 NO

Questions? [Contact us](#)

