**How to Add Timestamps to Core PDF Document Using External Signing**

The Syncfusion Essential® PDF is a comprehensive, high-performance [**.NET PDF library**](https://www.syncfusion.com/document-processing/pdf-framework/net-core) that enables you to create, read, and edit PDF documents programmatically without relying on Adobe dependencies. With this library, you can add timestamps to a PDF document using external signing in C#.

This section details the process of adding a timestamp during external signing. We provide a solution that generates a timestamp response from a timestamp server URI using the **[BouncyCastle](https://www.nuget.org/packages/Portable.BouncyCastle" \t "_blank)** library. The accepted timestamp token complies with the RFC3161 standard.

**Steps to Add Timestamps to a PDF Document Using External Signing Programmatically**

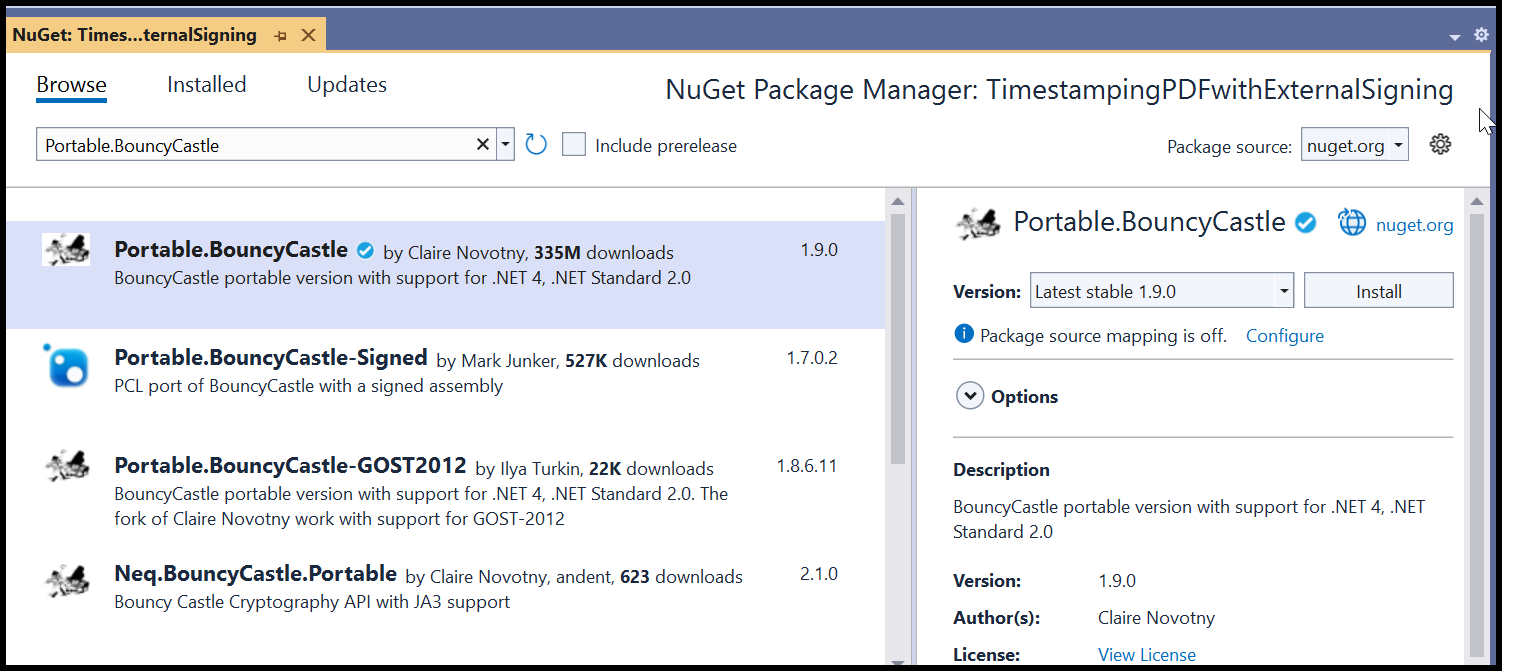
1. Create a new console application project.A screenshot of a computer

   AI-generated content may be incorrect.
2. Install the **[Syncfusion.Pdf.Net.Core](https://www.nuget.org/packages/Syncfusion.Pdf.Net.Core" \t "_blank)** NuGet package as a reference to your console application from [**Nuget.org**](https://www.nuget.org/).

A screenshot of a computer

AI-generated content may be incorrect.

1. Install the **[Portable.BouncyCastle](https://www.nuget.org/packages/Portable.BouncyCastle" \t "_blank)** NuGet package as a reference to your console application from [**Nuget.org**](https://www.nuget.org/).



In this example, we have used the open-source **[BouncyCastle](https://www.nuget.org/packages/Portable.BouncyCastle" \t "_blank)** library. Ensure you review its licensing before including it in your production environment. Alternatively, you can use the timestamp token provided by your service provider.

1. Include the following namespaces in the **Program.cs** file.

**C#**

**using** Syncfusion.Pdf.Parsing;

**using** Syncfusion.Pdf.Security;

**using** Syncfusion.Drawing;

**using** System.Security.Cryptography.X509Certificates;

**using** System.Security.Cryptography;

**using** Org.BouncyCastle.Tsp;

**using** System.Net;

**using** System.Text;

**using** Org.BouncyCastle.Math;

1. Use the following code sample in **Program.cs** to adding timestamps to a PDF document during the external signing process.

**C#**

// Get the stream from the input PDF document.

FileStream documentStream = **new** FileStream("Input.pdf", FileMode.Open, FileAccess.Read);

// Load the existing PDF document.

PdfLoadedDocument loadedDocument = **new** PdfLoadedDocument(documentStream);

// Create a digital signature for the first page of the document.

PdfSignature signature = **new** PdfSignature(loadedDocument, loadedDocument.Pages[0], null, "Signature");

// Set the signature bounds and cryptographic settings.

signature.Bounds = **new** RectangleF(**new** PointF(0, 0), **new** SizeF(100, 30));

signature.Settings.CryptographicStandard = CryptographicStandard.CADES;

signature.Settings.DigestAlgorithm = DigestAlgorithm.SHA1;

// Create an external signer using the SHA1 hash algorithm.

IPdfExternalSigner externalSignature = **new** ExternalSigner("SHA1");

// Add the public certificates for the external signer.

List<X509Certificate2> certificates = **new** List<X509Certificate2>();

certificates.Add(**new** X509Certificate2(Convert.FromBase64String(PublicCert)));

signature.AddExternalSigner(externalSignature, certificates, null);

//Create file stream.

**using** (FileStream outputFileStream = **new** FileStream(Path.GetFullPath(@"Output.pdf"), FileMode.Create, FileAccess.ReadWrite))

{

//Save the PDF document to file stream.

loadedDocument.Save(outputFileStream);

}

//Close the document.

loadedDocument.Close(true);

To sign the document using the **X509Certificate2** API and generate an RFC3161-compliant timestamp token with the **[BouncyCastle](https://www.nuget.org/packages/Portable.BouncyCastle" \t "_blank)** library, add the following code:

**C#**

// External signer class that implements IPdfExternalSigner for signing the document hash.

**class** **ExternalSigner** : **IPdfExternalSigner**

{

**private** string \_hashAlgorithm;

// Gets the hash algorithm used for signing.

**public** string HashAlgorithm

{

**get** { **return** \_hashAlgorithm; }

}

// Constructor that sets the hash algorithm for the signer.

**public** **ExternalSigner**(string hashAlgorithm)

{

\_hashAlgorithm = hashAlgorithm;

}

// Sign the document hash and return the timestamp response.

**public** byte[] **Sign**(byte[] message, **out** byte[] timeStampResponse)

{

byte[] signedBytes = null;

X509Certificate2 digitalID = **new** X509Certificate2(**new** X509Certificate2(Path.GetFullPath(@"Data/PDF.pfx"), "password123"));

// Use the appropriate signing algorithm based on the private key type.

**if** (digitalID.PrivateKey **is** System.Security.Cryptography.RSACryptoServiceProvider)

{

System.Security.Cryptography.RSACryptoServiceProvider rsa = (System.Security.Cryptography.RSACryptoServiceProvider)digitalID.PrivateKey;

signedBytes = rsa.SignData(message, HashAlgorithm);

}

**else** **if** (digitalID.PrivateKey **is** RSACng)

{

RSACng rsa = (RSACng)digitalID.PrivateKey;

signedBytes = rsa.SignData(message, System.Security.Cryptography.HashAlgorithmName.SHA1, RSASignaturePadding.Pkcs1);

}

// Generate an RFC3161 timestamp token for the signed data.

timeStampResponse = GetRFC3161TimeStampToken(signedBytes);

**return** signedBytes;

}

// Generate the RFC3161 timestamp token using the provided signed data.

**public** byte[] **GetRFC3161TimeStampToken**(byte[] bytes)

{

SHA1 sha1 = SHA1CryptoServiceProvider.Create();

byte[] hash = sha1.ComputeHash(bytes);

// Create a timestamp request using the SHA1 hash.

TimeStampRequestGenerator reqGen = **new** TimeStampRequestGenerator();

reqGen.SetCertReq(true);

TimeStampRequest tsReq = reqGen.Generate(TspAlgorithms.Sha1, hash, BigInteger.ValueOf(100));

byte[] tsData = tsReq.GetEncoded();

// Send the timestamp request to the server.

HttpWebRequest req = (HttpWebRequest)WebRequest.Create("https://rfc3161.ai.moda"); // Update your timestamp URI here

req.Method = "POST";

req.ContentType = "application/timestamp-query";

req.Headers.Add("Authorization", "Basic " + Convert.ToBase64String(Encoding.ASCII.GetBytes("9024:yourPass")));

req.ContentLength = tsData.Length;

// Write the request data to the stream.

Stream reqStream = req.GetRequestStream();

reqStream.Write(tsData, 0, tsData.Length);

reqStream.Close();

// Get the timestamp response from the server.

HttpWebResponse res = (HttpWebResponse)req.GetResponse();

**if** (res != null)

{

Stream resStream = **new** BufferedStream(res.GetResponseStream());

TimeStampResponse tsRes = **new** TimeStampResponse(resStream);

**return** tsRes.TimeStampToken.GetEncoded();

}

**return** null;

}

}

A complete working sample can be downloaded from [**GitHub**](https://github.com/SyncfusionExamples/PDF-Examples/tree/master/Digital%20Signature/TimestampingPDFwithExternalSigning/).  
By executing the program, you will get the PDF document as follows.

A screenshot of a computer

AI-generated content may be incorrect.

Take a moment to peruse the [**documentation**](https://help.syncfusion.com/document-processing/pdf/pdf-library/net/working-with-digitalsignature#adding-timestamps-to-a-pdf-document-using-external-signing) to add timestamps to a PDF document using external signing.

Refer [**here**](https://www.syncfusion.com/document-processing/pdf-framework/net) to explore the rich set of Syncfusion Essential® PDF features.

**Conclusion**  
I hope you enjoyed learning on how to add timestamps to Core PDF document using external signing.

You can refer to our [**ASP.NET Core PDF feature tour**](https://www.syncfusion.com/document-processing/pdf-framework/net-core) page to know about its other groundbreaking feature representations and [**documentation**](https://help.syncfusion.com/file-formats/pdf/create-pdf-file-in-asp-net-core), and how to quickly get started for configuration specifications. You can also explore our [**ASP.NET Core PDF example**](https://ej2.syncfusion.com/aspnetcore/PDF/Default#/bootstrap5) to understand how to create and manipulate data.

For current customers, you can check out our components from the [**License and Downloads**](https://www.syncfusion.com/sales/teamlicense) page. If you are new to Syncfusion®, you can try our 30-day [**free trial**](https://www.syncfusion.com/downloads/aspnetcore-js2) to check out our other controls.

If you have any queries or require clarifications, please let us know in the comments section below. You can also contact us through our [**support forums**](https://www.syncfusion.com/forums), [**Direct-Trac**](https://support.syncfusion.com/create), or [**feedback portal**](https://www.syncfusion.com/feedback/aspnet-core?control=pdf). We are always happy to assist you!