



Web



Signature lifecycle



WEB > GUIDES > PSPDFKIT SERVER > DIGITAL SIGNATURES > SIGNATURE LIFECYCLE

How to implement digital signatures in PDFs



PSPDFKit Server has been deprecated and replaced by [Document Engine](#). To migrate to Document Engine and unlock advanced document processing capabilities, refer to our [migration guide](#). Learn more about these enhancements on our [blog](#).

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

- 1 PSPDFKit Server prepares the document for a signature, adding an invisible form field that will contain the signature value.
- 2 PSPDFKit Server then contacts an **external signing service** you're responsible for setting up, which will provide a compliant signature value.
- 3 PSPDFKit Server applies the returned signature to the document and saves it, storing the final file as an asset associated with the document and the used Instant layer.

This architecture ensures that PSPDFKit 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.

The signing service

The signing service is a network service that you're responsible for maintaining and operating.



ASK AI

It needs to expose a single HTTP endpoint of your choice that receives all the information required to calculate a compliant digital signature, and it should return a DER PKCS#7 container that can be set as a value of the digital signature field.

For example, say you want to sign a document with the ID `my-document-id` via the Server API:

```
1 POST http://localhost:5000/api/documents/my-document-id/sign
2 Authorization: Token token="<secret token>"
3 Content-Type: application/json
4
5 {
6   "signingToken" : "custom-token"
7 }
```

The request accepts an optional `signingToken` string parameter, which will be forwarded to the signing service in the exact same shape.

You can use it to pass a token that can be used to verify the authenticity of the signing request or to provide identity information about the user applying the signature.

The signing endpoint will receive a request with the following schema:

```
1 POST http://signing-server:6000/sign
2 Content-Type: application/json
3
4 {
5   "encoded_contents" : "CkVudW11cmF0aW5nIG9iamVjdHM6IDExLCBkb25lLgpDb3VudGluZy!"
6   "digest" : "aab7fe5d814e7e8048275d19693435013727ee8002b85ba8edc29321fc2edfc9"
7   "signing_token" : "custom-token"
8 }
```

In the example above, we assume that the signing service can be accessed at `http://signing-server:6000/sign`.

The endpoint will receive a JSON-encoded `POST` request containing:

- ⌘ The **Base64-encoded contents** of the file to sign. This represents the portion of the PDF document covered by the digital signature, minus the byte range that will contain the signature itself. Note that since it's Base64-encoded, you'll need to decode it before signing it.
- ⌘ The **digest** for the contents to be signed (with the hash calculated before the contents are encoded to Base64). If your language and encryption libraries support it, you can perform the

signature operation using the hash as the signature contents. In such a case, make sure you configure PSPDFKit Server to use at least `sha256` as its hashing algorithm.

✦ The **signing token**, forwarded from the previous step.

For more details, refer to [our signing service reference implementation](#) on GitHub.

We recommend setting up the signing service as a container on the same network as PSPDFKit Server and without external network access to guarantee fast, consistent performance and better security.

Once the signing service is up and running, you can configure PSPDFKit Server to use it by setting the `SIGNING_SERVICE_URL` to the signing service endpoint, e.g. `http://signing-service:6000/sign`. For more information on configuration and customization, refer to our [configuration](#) guide.

Applying a signature

To digitally sign a document, perform a call to the `PSPDFKit.Instance#signDocument` method. As its first argument, during the signing preparation of the document for the PKCS#7 container, you can optionally pass an object with a `placeholderSize` property that can be used to override the default size that's reserved for the signature.

As a second argument, you can optionally specify an object with the `signingToken` string property that was described in the [previous subsection](#). Refer to the [API documentation](#) for more details:

```
1 instance
2   .signDocument(null, {
3     signingToken: "user-1-with-rights"
4   })
5   .then(() => {
6     console.log("document signed.");
7   })
8   .catch((error) => {
9     console.error("The document could not be signed.", error);
10  });
```

Was this helpful?

✓ YES

✗ NO

