



# ENTRUST

## Entrust Signing Automation Service User Guide

**Document issue: 1.3**  
**Issue date: October 16, 2024**

© 2024, Entrust. All rights reserved

*Entrust and the hexagon design are trademarks, registered trademarks and/or service marks of Entrust Corporation in Canada and the United States and in other countries. All Entrust product names and logos are trademarks, registered trademarks and/or service marks of Entrust Corporation. All other company and product names and logos are trademarks, registered trademarks and/or service marks of their respective owners in certain countries.*

*This information is subject to change as Entrust reserves the right to, without notice, make changes to its products as progress in engineering or manufacturing methods or circumstances may warrant. The material provided in this document is for information purposes only. It is not intended to be advice. You should not act or abstain from acting based upon such information without first consulting a professional. ENTRUST DOES NOT WARRANT THE QUALITY, ACCURACY OR COMPLETENESS OF THE INFORMATION CONTAINED IN THIS ARTICLE. SUCH INFORMATION IS PROVIDED "AS IS" WITHOUT ANY REPRESENTATIONS AND/OR WARRANTIES OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, BY USAGE OF TRADE, OR OTHERWISE, AND ENTRUST SPECIFICALLY DISCLAIMS ANY AND ALL REPRESENTATIONS, AND/OR WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT, OR FITNESS FOR A SPECIFIC PURPOSE.*

## Contents

<b>1 About this guide .....</b>	<b>5</b>
Acronyms.....	5
Revision information.....	9
Other documents .....	9
Documentation feedback .....	9
<b>2 Service licensing.....</b>	<b>10</b>
<b>3 Benefits.....</b>	<b>11</b>
<b>4 Architecture.....</b>	<b>12</b>
Customer premises .....	12
Entrust datacenters .....	12
Entrust Signing Automation user .....	13
Service rate.....	13
Signing certificate .....	13
Signing REST API .....	13
Virtual Token .....	15
<b>5 Managing tokens with Entrust Certificate Services cloud .....</b>	<b>16</b>
Creating a Virtual Token .....	16
Creating a document signing certificate.....	17
Creating a code signing certificate.....	20
Getting API credentials .....	22
<b>6 Signing Windows files with the Entrust KSP library .....</b>	<b>25</b>
Entrust KSP library requirements .....	25
Downloading the Entrust KSP library .....	25
Installing the Entrust KSP library .....	27
Installing and configuring Windows SignTool.....	28
Signing Windows files with SignTool and the Entrust KSP library .....	29
<b>7 Signing with the Entrust PKCS #11 library.....</b>	<b>31</b>
Installing the PKCS #11 library with the Signing Automation Client.....	31

Integrating the PKCS #11 library with third-party applications .....	49
<b>8 Signing with REST clients .....</b>	<b>63</b>
Signing data with curl commands .....	63
Signing data with Postman .....	67
Swagger API reference .....	75
<b>9 Debugging the Signing Automation Client .....</b>	<b>76</b>
Enabling the debugging mode .....	76
Disabling the debugging mode .....	77

# 1 About this guide

This document provides a complete customer guide for the Entrust Signing Automation Service (SAS).

- [Acronyms](#)
- [Revision information](#)
- [Other documents](#)
- [Documentation feedback](#)

## Acronyms

See below a definition of acronyms that may appear in this document.

Acronym	Definition
ACME	Automatic Certificate Management Environment
ADCS	Microsoft Active Directory Certificate Services
ADDS	Microsoft Active Directory Domain Services
AES	Advanced Encryption Standard
AIA	Authority Information Access
CA	Certification Authority
CAGW	Entrust CA Gateway (API)
CEG	Entrust Certificate Enrollment Gateway
CEP	Certificate Enrollment Policy
CLI	Command-line Interface
CLM	Certificate Lifecycle Management
CMC	Cryptographic Message Syntax
CMP	Certificate Management Protocol
CN	Common Name

<b>Acronym</b>	<b>Definition</b>
CPS	Certification Practice Statement
CRL	Certificate Revocation List
CSR	Certificate Signing Request (PKCS #10)
CSS	Certificate Status Server
CT	Certificate Transparency
DHCP	Dynamic Host Configuration Protocol
DN	Distinguished Name
DNS	Domain Name System
ECDSA	Elliptic Curve Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
ECS	Entrust Certificate Services
EEE	End Entity Enrollment
EST	Enrolment over Secure Transport
FIPS	Federal Information Processing Standard
FQDN	Fully Qualified Domain Name
JDK	Java Development Kit
HSM	Hardware Security Module
LDAP	Lightweight Directory Access Protocol
LDAPS	Lightweight Directory Access Protocol over SSL

Acronym	Definition
LRA	Local Registration Authority
MDM	Mobile Device Management
MDMWS	Entrust's Mobile Device Management Web Service API
MS-XCEP	X.509 Certificate Enrollment Policy Protocol (CEP)
MS-WSTEP	WS-Trust X.509v3 Token Enrollment Extensions Protocol (WSTEP)
NIST	National Institute of Standards and Technology
PKIaaS	Public Key Infrastructure as a Service
OA	Operational Authority
OCSP	Online Certificate Status Protocol
OID	Object Identifier
OTP	One-time Passcode
OVA	Open Virtual Appliance
P12	PKCS (Public Key Cryptography Standards) #12
PA	Policy Authority
PQ	Post-Quantum
PKCS	Public Key Cryptography Standards
PKI	Public Key Infrastructure
RA	Registration Authority
REST	Representational State Transfer

<b>Acronym</b>	<b>Definition</b>
RBAC	Role-Based Access Control
RDN	Relative Distinguished Name
RFC	Request for Comment
RHEL	Red Hat Enterprise Linux
RPO	Recovery Point Objective
RTO	Recovery Time Objective
SAN	Subject Alternative Names
SCEP	Simple Certificate Enrollment Protocol
SIEM	Security Information and Event Management
SHA	Secure Hash Algorithms
S/MIME	Secure/Multipurpose Internet Mail Extensions
TLS	Transport Layer Security
TPM	Trusted Platform Module
URL	Uniform Resource Locator
UEM	Unified Endpoint Management
V2G	Vehicle-to-Grid
VM	Virtual Machine
WHFB	Windows Hello for Business



## Revision information

See the below table for the issues of this document.

Issue	Date	Section	Changes
1.3	Oct 2024	<a href="#">Service licensing</a>	Add license agreement for the Entrust KSP library
1.2	Oct 2024	<a href="#">Signing Windows files with the Entrust KSP library</a>	New section
1.1	Jun 2023	<a href="#">Signing with REST clients</a>	New section
1.0	Apr 2022	All sections	Release the document

## Other documents

See the table below for other relevant documentation on the Entrust Signing Automation Service (SAS).

Document	URL
Entrust SAS product page	<a href="https://www.entrust.com/products/digital-signing/digital-signing-as-a-service/signing-automation-service">https://www.entrust.com/products/digital-signing/digital-signing-as-a-service/signing-automation-service</a>
Digital signing terms and conditions	<a href="https://www.entrust.com/legal-compliance/terms-conditions/digital-signing">https://www.entrust.com/legal-compliance/terms-conditions/digital-signing</a>

## Documentation feedback

You can rate and provide feedback about product documentation by completing the online feedback form:

<https://go.entrust.com/documentation-feedback>

Any information you provide goes directly to the documentation team and is used to improve and correct the information in our guides.

## 2 Service licensing

This section defines the licensing terms and permitted uses for the Signing Automation Service, including the Signing Client software.

In this document, when we refer to “you”, we mean the customer who has purchased the Signing Automation Service or one of that customer’s Users. In general, the Signing Automation Service is for the customer’s internal use only. However, the customer is permitted to appoint outside organizations and/or their software applications as Users and distribute the Signing Client software to such Users, solely to enable those Users to use the Signing Automation Service on the customer’s behalf (and not for the outside User’s benefit). The customer is responsible for overseeing and controlling how the Users use the Signing Automation Service.

You may deploy the Entrust Signing Client software on your company’s infrastructure and/or commercial cloud accounts. You are strongly encouraged to keep your deployments up to date with our latest product release.

You may use the Signing Automation Service only to apply for and in connection with one or more Digital Certificates that identify you (Entrust’s customer) as the Subject by your organization name (e.g., “ABC Ltd”) or one of your corporate affiliates if that affiliate authorizes you to digitally sign hashed data on its behalf. You are expressly prohibited from using the Signing Automation Service together with Digital Certificates that identify as the Subject any person other than yourself or an affiliate that has authorized you to sign hashed data on its behalf. You are expressly prohibited from using the Signing Automation Service in conjunction with any Digital Certificate not issued by Entrust or other Entrust PKI services.

For each subscription to the Signing Automation Service, we will provide a license key that determines the number of digital signatures that you may generate using the service, the number of Digital Certificates with which you may use the service, and/or for otherwise enabling or controlling certain functions within the service. You may not copy or alter your license key. You may not circumvent or attempt to circumvent the license key mechanism. You may only use a license key provided by Entrust and only in conjunction with the Signing Automation Service and Signing Client software and its APIs for which it was delivered. Unless otherwise specified in your Order(s), each Entrust Signing Automation Service license key is capped to one Digital Certificate based and a number of signatures to be consumed within one year. Additional Digital Certificates or number of signatures can be purchased separately.

Your subscription to the Signing Automation Service provides you with access to Entrust Timestamping Authority service, which permits you to obtain the same number of timestamps as the number of contracted signatures within the same period.

Each issuing or reissuing of a signing certificate consumes one of the 10,000 licensed signatures.

The Entrust Signing Client software includes software developed by the OpenSSL Software Foundation, and provides an interface compatible with the PKCS #11 Cryptographic Token Interface (Cryptoki) by RSA Security Inc. and Microsoft KSP/CNG Key Storage Provider. Please review the software attributes for the full software list.

### 3 Benefits

The Entrust Signing Automation Service is a cloud-based service that allows developers and organizations to securely store and digitally sign code and applications with a trusted certificate authority (CA). Specifically, this solution:

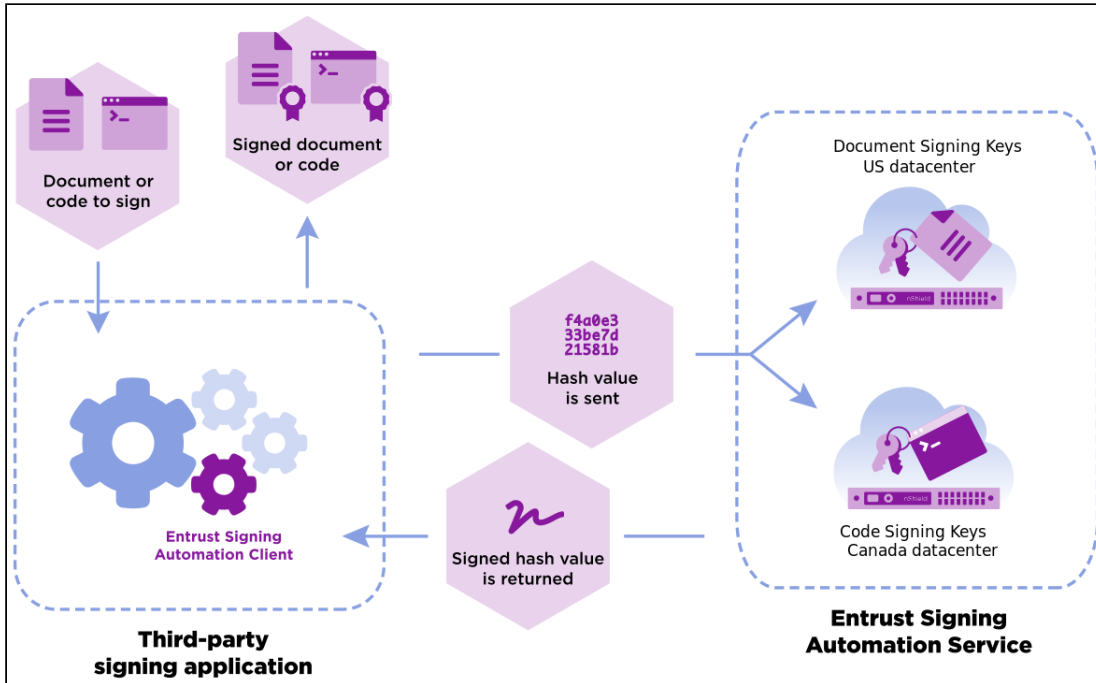
- Secures private keys in accordance with the baseline requirements for FIPS 140-2 Level 3 certified key storage.
- Automates signing software.
- Integrates seamlessly with your existing tools and development workflow.
- Is fully compatible with the Entrust PKI Cloud Suite.

See the following table for the file formats supported by each integration use case.

Integration use case	File format
<a href="#">Signing Windows files with the Entrust KSP library</a>	Windows, Microsoft Installers (MSI), Cabinet files (CAB), Catalog files (CAT), Windows packages (APPX/MSIX), Microsoft Dynamics 365 extension packages, NuGet packages and scripts (PowerShell, VBScript, JScript, WSF)
<a href="#">Signing PDF documents with iText</a>	PDF
<a href="#">Signing Microsoft Authenticode files with Jsign</a>	Windows, Microsoft Installers (MSI), Cabinet files (CAB), Catalog files (CAT), Windows packages (APPX/MSIX), Microsoft Dynamics 365 extension packages, NuGet packages and scripts (PowerShell, VBScript, JScript, WSF)
<a href="#">Signing Java files with Jarsigner</a>	Java CAB, EAR, JAR, SAR, and WAR files.
<a href="#">Signing with REST clients</a>	Generic hashes


## 4 Architecture

The Entrust Signing Automation Service is a cloud-based hash signing service, with private keys securely generated and protected by the service. Using this service, you can sign hashes generated on documents or code files.



See below for a description of the main concepts.

- [Customer premises](#)
- [Entrust datacenters](#)
- [Entrust Signing Automation user](#)
- [Service rate](#)
- [Signing certificate](#)
- [Signing REST API](#)
- [Virtual Token](#)

 The Entrust Signing Automation Service includes timestamping and OCSP validation services.

### Customer premises

The customer premises run:

- The Entrust Signing Automation Client.
- The signing application: A REST client or a third-party product with signing capabilities such as the iText library, Oracle JDK, OpenJDK, or Entrust Java Toolkits.

### Entrust datacenters

In the cloud-based Entrust data centers, the multi-tenant signing key management selects the nShield HSM that will:

- Generate and wrap the keys when processing key generation requests.
- Unwrap the signing key of the signing key database and sign a document hash when processing a signature request.

nShield HSMs are FIPS 140-2 L3 compliant devices for:

- Generating and wrapping the keys stored in the signing key database.
- Unwrapping the signing keys.
- Signing document hashes with the unwrapped keys.

**i** The Entrust data centers never access the documents to be signed because generating a digital signature only requires the document hash.

## Entrust Signing Automation user

In the Entrust Certificate Service portal, the users of the Signing Automation Service are referred to as "Signing Clients". All service users have the same functionality when consuming the Signing Automation Service from computers configured for that purpose. However, it will be usual to separate the following roles.

Role	Action
Administrator	Create signing keys and import certificates in the token, as explained in <a href="#">Managing tokens with Entrust Certificate Services cloud</a> .
Software application	Select a token key and sign documents.

## Service rate

Entrust Signing Automation Service can deliver a minimum of 10 signatures per second and customer.

## Signing certificate

Depending on the data you want to sign, you must either purchase:

- A Document Signing Certificate
- A Code Signing Certificate

See [Managing tokens with Entrust Certificate Services cloud](#) for how to obtain these certificates.

## Signing REST API

As explained [Signing with REST clients](#), you can use the Signing Automation Client to consume the Entrust Signing Automation Service. See below for a description of the main related concepts.

- [Authentication token](#)
- [Identity Provider service](#)
- [Raw Signature service](#)
- [User partition](#)

## Authentication token

With the credentials described in [Getting API credentials](#), the [Identity Provider service](#) provides authentication tokens for registered users. Each authentication token:

- Is the Base64 encoding of a JSON Web Token (JWT).
- Allows a user to consume the [Raw Signature service](#) for up to two minutes.

## Identity Provider service

The Identity Provider service of the Entrust Signing Automation Service API manages users and credentials of the [Raw Signature service](#). Specifically, this service grants users [Authentication tokens](#) to consume the [Raw Signature service](#).

As detailed in the table below, the URL and IP of this service depend on the purchased Entrust signing service.

Purchased service	IdP service URL	IdP service IP
Entrust Code Signing as a Service	<a href="https://idp.csaas.entrust.net">https://idp.csaas.entrust.net</a>	72.140.233.100
Entrust Document Signing as a Service	<a href="https://idp.pkiaas.entrust.com">https://idp.pkiaas.entrust.com</a>	54.81.74.253 52.2.29.234

 See <https://api.managed.entrust.com/sas/idp-api> for the Swagger reference of this service.

## Raw Signature service

The Raw Signature service of the Signing Automation Service API:

- Manages the user keys and certificates, grouped in [Virtual Tokens](#).
- Generates signatures on raw data – typically, the digest of a document.

As detailed in the table below, the URL and IP of this service depend on the purchased Entrust signing service.

Purchased service	Raw Signature service URL	Raw Signature service IP
Entrust Code Signing as a Service	<a href="https://rawsigner.csaas.entrust.net">https://rawsigner.csaas.entrust.net</a>	72.140.233.100
Entrust Document Signing as a Service	<a href="https://rawsigner.pkiaas.entrust.com">https://rawsigner.pkiaas.entrust.com</a>	54.81.74.253 52.2.29.234

 See <https://api.managed.entrust.com/sas/rawsigner-api> for the Swagger reference of this service.

## User partition

In the Signing Automation Service API, the user *partition* refers to the user organization.

## Virtual Token

The Signing Automation Service uses “virtual tokens” as per the “physical” tokens described in the PKCS #11 standard. When your signing application connects to the server, the Signing Automation Client provides a virtual token with the following standard operations.

<b>PKCS #11 Mechanism</b>	<b>Supported operations</b>
CKM_ECDSA	Sign and verify with NIST P256, NIST P384, or NIST P521 curves.
CKM_ECDSA_KEY_PAIR_GEN	Generate ECDSA key pairs with NIST P256, NIST P384, or NIST P521 curves.
CKM_RSA_PKCS	Sign and verify with PKCS1.5 padding (and software-based hashing) with 2048, 3072, or 4096 key sizes.
CKM_RSA_PKCS_KEY_PAIR_GEN	Generate RSA key pairs with 2048, 3072, or 4096 key sizes.
CKM_SHA256_RSA_PKCS	Sign and verify with PKCS1.5 and SHA 256 hashing with 2048, 3072, or 4096 key sizes.
CKM_SHA384_RSA_PKCS	Sign and verify with PKCS1.5 and SHA 384 hashing with 2048, 3072, or 4096 key sizes.
CKM_SHA512_RSA_PKCS	Sign and verify with PKCS1.5 and SHA 512 hashing with 2048, 3072, or 4096 key sizes.

## 5 Managing tokens with Entrust Certificate Services cloud

Create and manage tokens, signing keys, certificates, and credentials in the Entrust Certificate Services cloud.

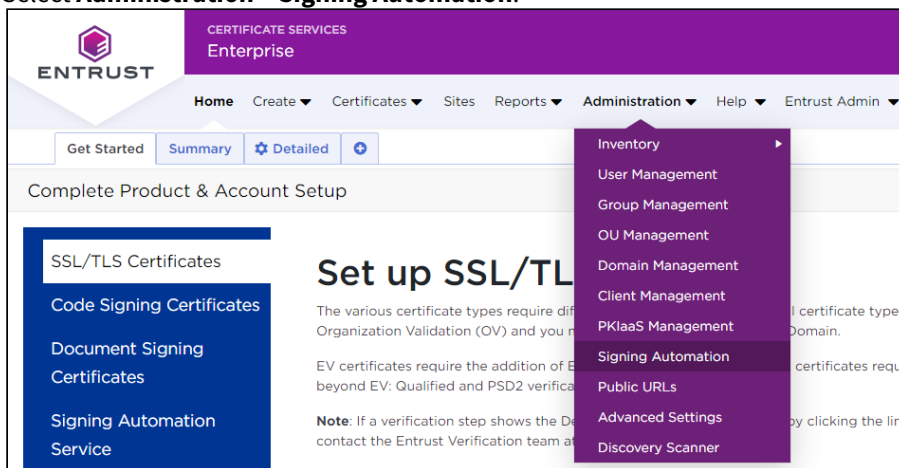
- [Creating a Virtual Token](#)
- [Creating a document signing certificate](#)
- [Creating a code signing certificate](#)
- [Getting API credentials](#)

### Creating a Virtual Token

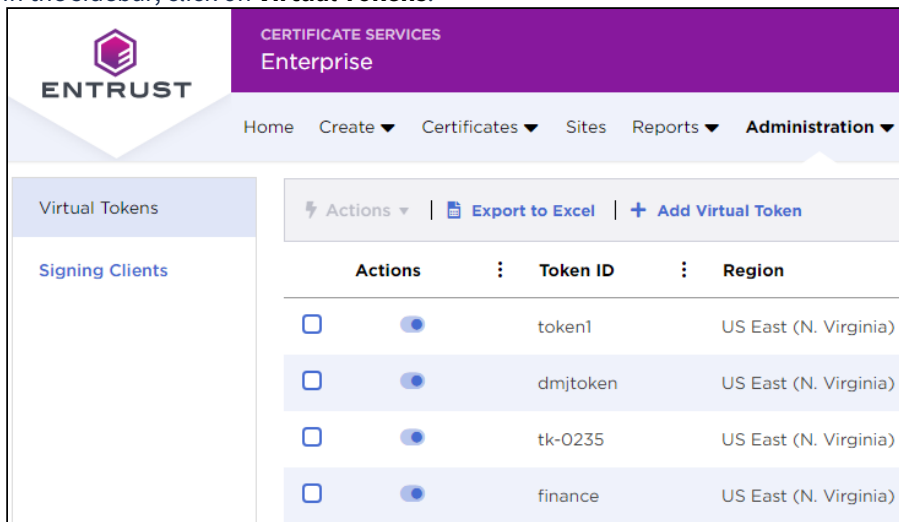
In the Entrust Certificate Services web portal, create at least one Virtual token to manage signing keys and certificates.

#### To create a Virtual Token in Entrust Certificate Services

1. Log in to the Entrust Certificate Services web portal at [cloud.entrust.net](https://cloud.entrust.net)
2. Select **Administration > Signing Automation**.

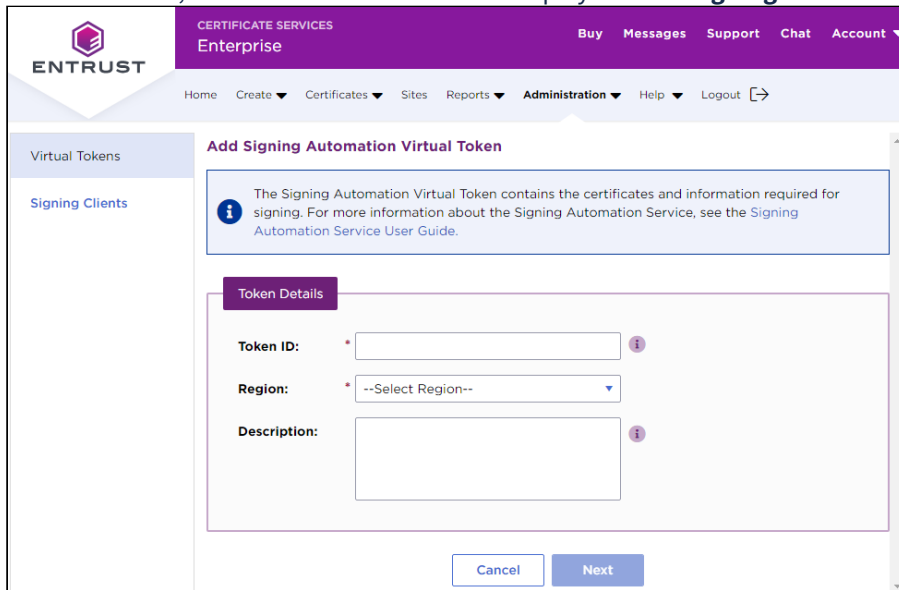


3. In the sidebar, click on **Virtual Tokens**.






4. In the menu bar, click **+ Add Virtual Token** to display the **Add Signing Automation Virtual Token** form.



5. Enter the following values.
  - [Token ID](#)
  - [Region](#)
  - [Description field](#)
6. Click **Next**.
7. Click **Submit** to confirm the Virtual Token creation.

## Token ID

A 3-14 character name to uniquely identify the new Virtual Token.

 When processing the Signing Automation License, the Signing Client application will display this identifier in the Virtual Token selector.

## Region

The region for storing the token keys. Select:

- **Code Signing Compliance (Canada)** for storing code signing keys.
- **US EAST (N. Virginia)** for storing document signing keys.

## Description field

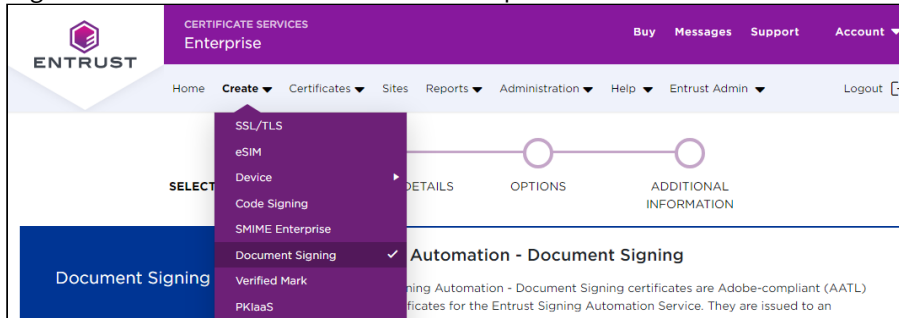
An optional description of the Virtual Token.

## Creating a document signing certificate

Document signing requires a certificate specific to that purpose.

### To create a document signing certificate

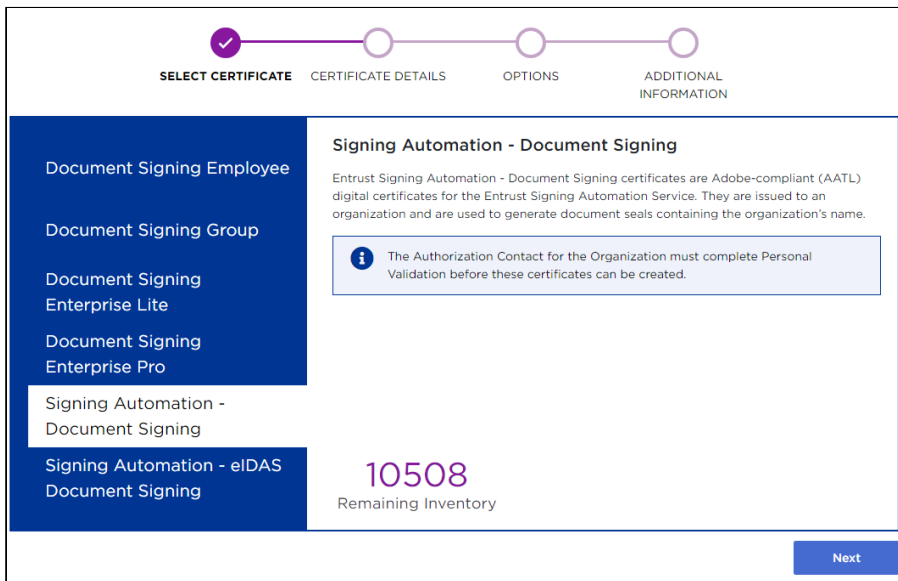
1. Log in to the Entrust Certificate Services web portal at [cloud.entrust.net](https://cloud.entrust.net).



2. In the **Create** menu, select **Document Signing**.
3. Follow the instructions on each page of the certificate creation wizard.
  - [SELECT CERTIFICATE](#)
  - [CERTIFICATE DETAILS](#)
  - [OPTIONS](#)
  - [ADDITIONAL INFORMATION](#)

## SELECT CERTIFICATE

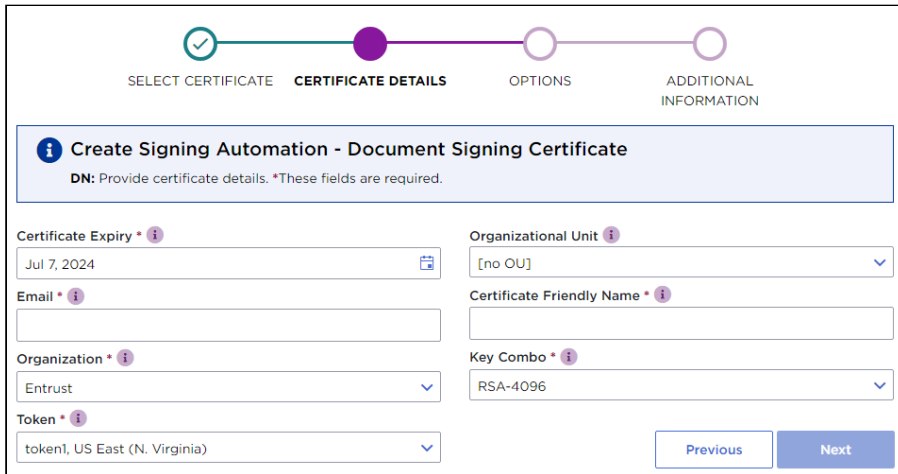
In the sidebar menu, click **Signing Automation - Document Signing**.



In the **Remaining Inventory** section of this page, check the number of certificates you can create. Click **Next** if you still have remaining certificates.

## CERTIFICATE DETAILS

Configure the new document signing certificate.



See the table below for a description of each field.

Field	Value	Mandatory
Certificate Expiry	The end date of the certificate validity.	✓
Email	The email address of the certificate owner. Entrust will use the selected email address to send notifications on the certificate status.	✓
Organization	The organization of the certificate owner.	✓
Organizational Unit	The unit within the organization (if any)	
Token	The <a href="#">Virtual Token</a> described in <a href="#">Creating a Virtual Token</a> .	✓
Certificate Friendly Name	A label for the keystores to identify the certificate when <a href="#">Integrating the PKCS #11 library with third-party applications</a> . Spaces or special characters may cause errors.	✓
Key Combo	The type and length of the certificate public key.	✓

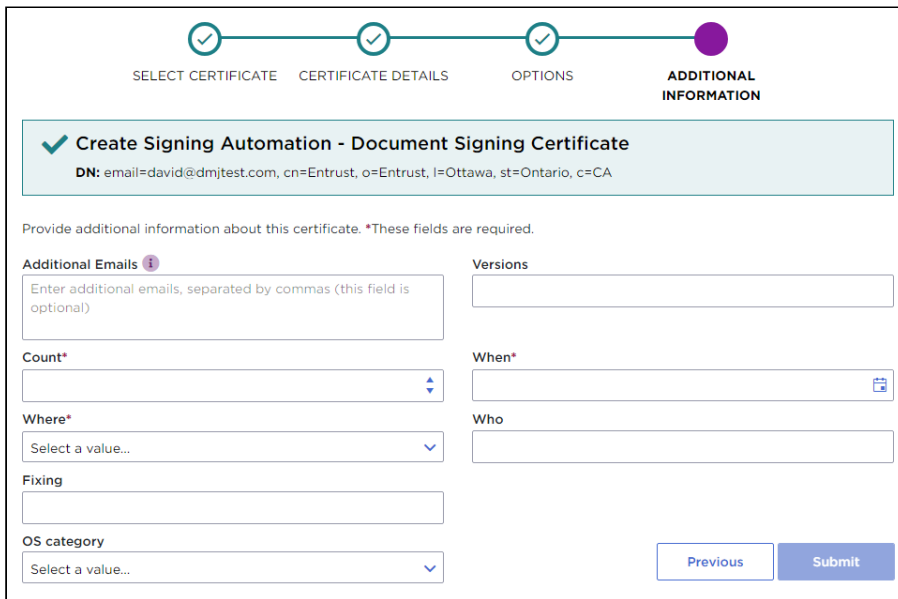
Click **Next**.

## OPTIONS

This page does not display additional options for the document signing certificates. Click **Next**.

## ADDITIONAL INFORMATION

Configure additional settings selected by your certificate services administrator.



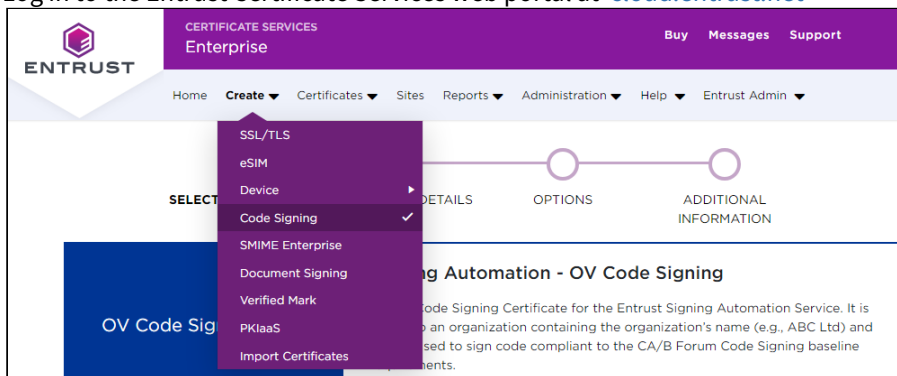
Click **Submit** and confirm the certificate creation.

## Creating a code signing certificate

Code signing requires a certificate specific to that purpose.

### To create a code signing certificate

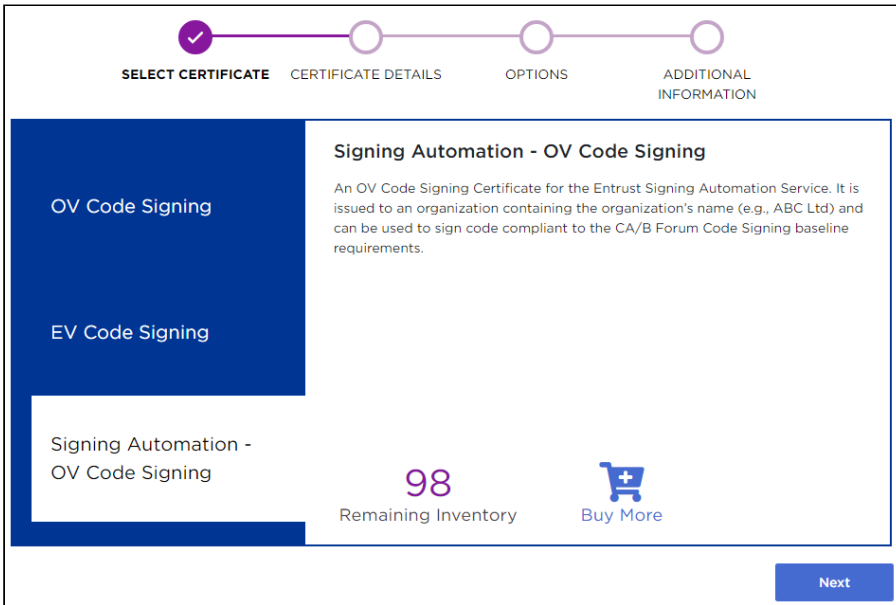
1. Log in to the Entrust Certificate Services web portal at [cloud.entrust.net](https://cloud.entrust.net)



2. Select **Create > Code Signing**.
3. Follow the instructions on each page of the certificate creation wizard.
  - [SELECT CERTIFICATE](#)
  - [CERTIFICATE DETAILS](#)
  - [OPTIONS](#)
  - [ADDITIONAL INFORMATION](#)

## SELECT CERTIFICATE

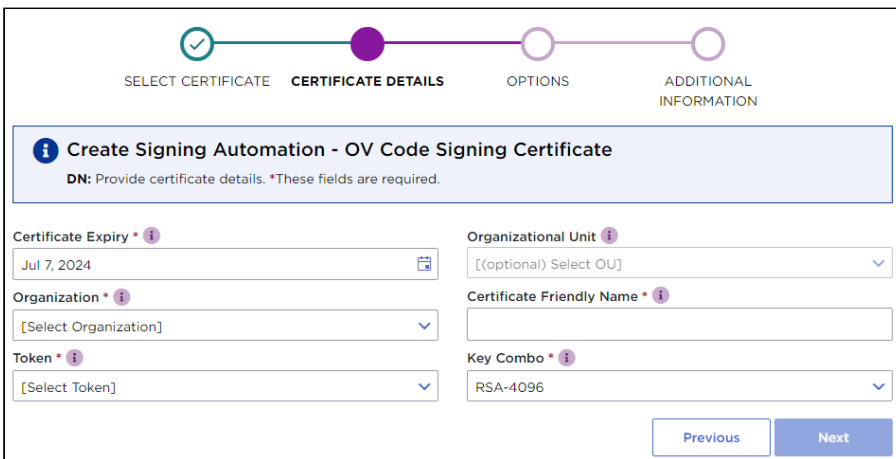
In the sidebar menu, click **Signing Automation - OV Code Signing**.



In the **Remaining Inventory** section of this page, check the number of certificates you can create. Click **Next** if you still have remaining certificates.

## CERTIFICATE DETAILS

Configure the new document signing certificate.



See the table below for a description of each field.

Field	Value	Mandatory
Certificate Expiry	The end date of the certificate validity.	✓

Field	Value	Mandatory
Organization	The organization of the certificate owner.	✓
Organization Unit	The unit within the organization (if any).	
Token	The <a href="#">Virtual Token</a> obtained in <a href="#">Creating a Virtual Token</a> .	✓
Certificate Friendly Name	A label for the keystores to identify the certificate when <a href="#">Integrating the PKCS #11 library with third-party applications</a> . Spaces or special characters may cause errors.	✓
Key Combo	The type and length of the certificate public key.	✓

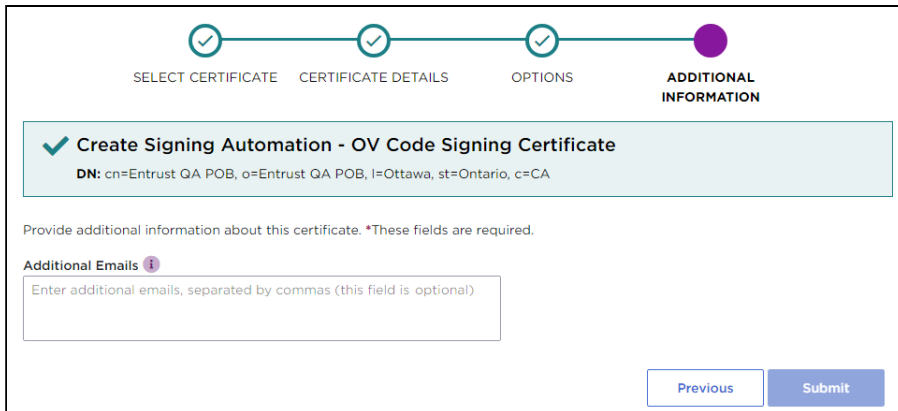
Click **Next**.

## OPTIONS

This page does not display additional options for the document signing certificates. Click **Next**.

## ADDITIONAL INFORMATION

Optionally, enter **Additional Emails** for receiving notifications on the certificate status.



The screenshot shows a progress bar with four steps: SELECT CERTIFICATE, CERTIFICATE DETAILS, OPTIONS, and ADDITIONAL INFORMATION. The 'ADDITIONAL INFORMATION' step is currently active and highlighted in purple. Below the progress bar, there is a section titled 'Create Signing Automation - OV Code Signing Certificate' with a green checkmark and a Distinguished Name (DN) field containing 'DN: cn=Entrust QA POB, o=Entrust QA POB, l=Ottawa, st=Ontario, c=CA'. Below this, there is a text input field for 'Additional Emails' with a help icon and a note that these fields are required. At the bottom right, there are 'Previous' and 'Submit' buttons.

Click **Submit** to confirm the certificate creation.

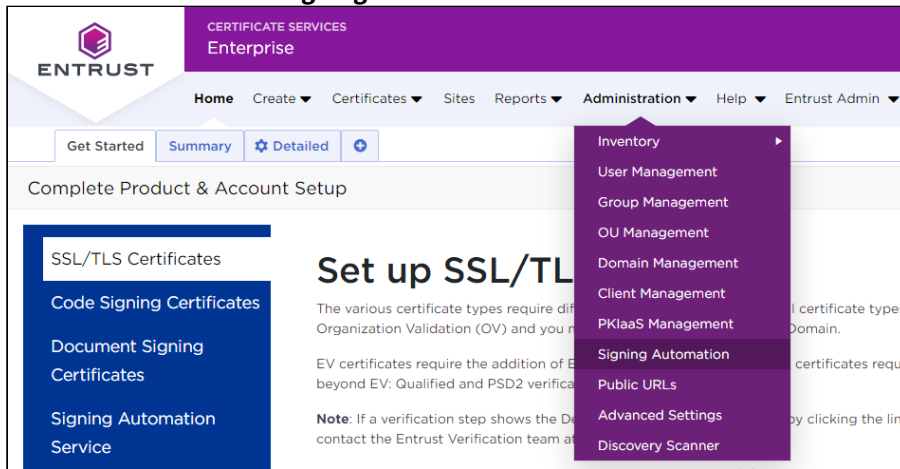
## Getting API credentials

See below for obtaining API credentials for consuming the Signing Automation Service with client applications.

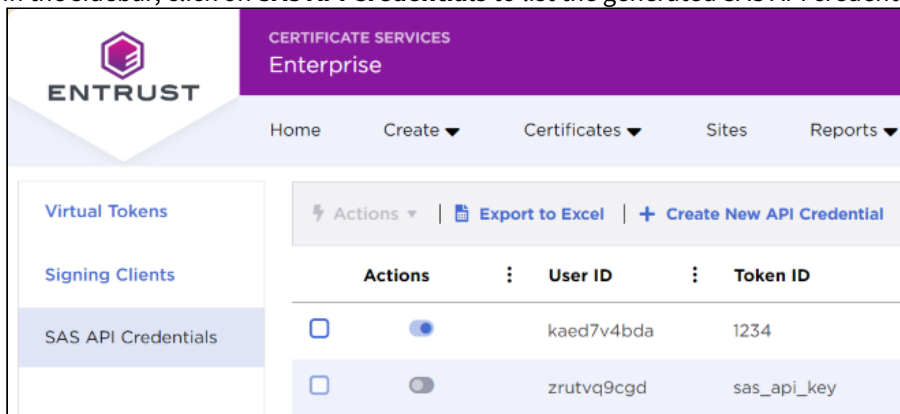
### To obtain API credentials

1. Log in to the Entrust Certificate Services web portal at [cloud.entrust.net](https://cloud.entrust.net)

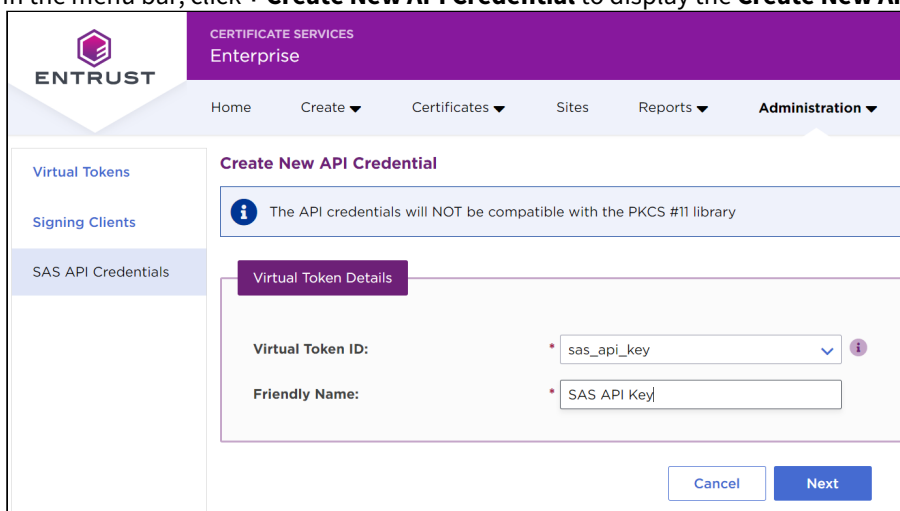
2. Select **Administration > Signing Automation**.



3. In the sidebar, click on **SAS API Credentials** to list the generated SAS API credentials.



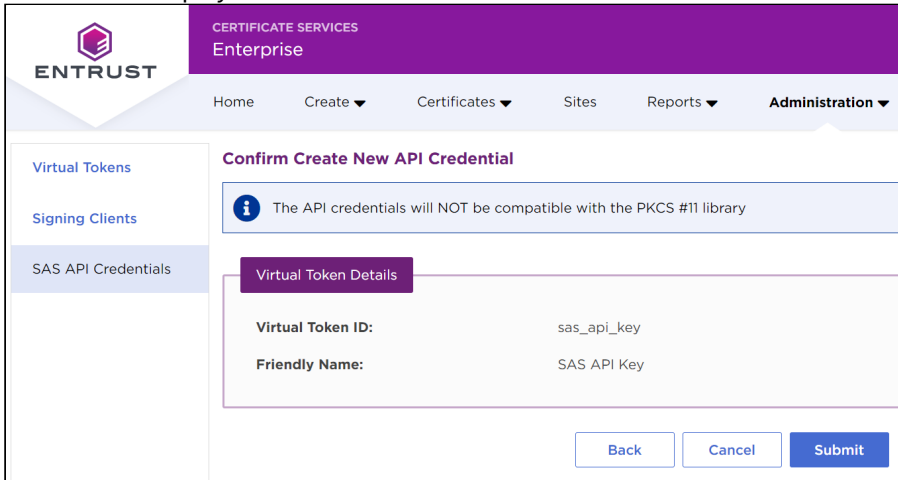
4. In the menu bar, click **+ Create New API Credential** to display the **Create New API Credential** form.



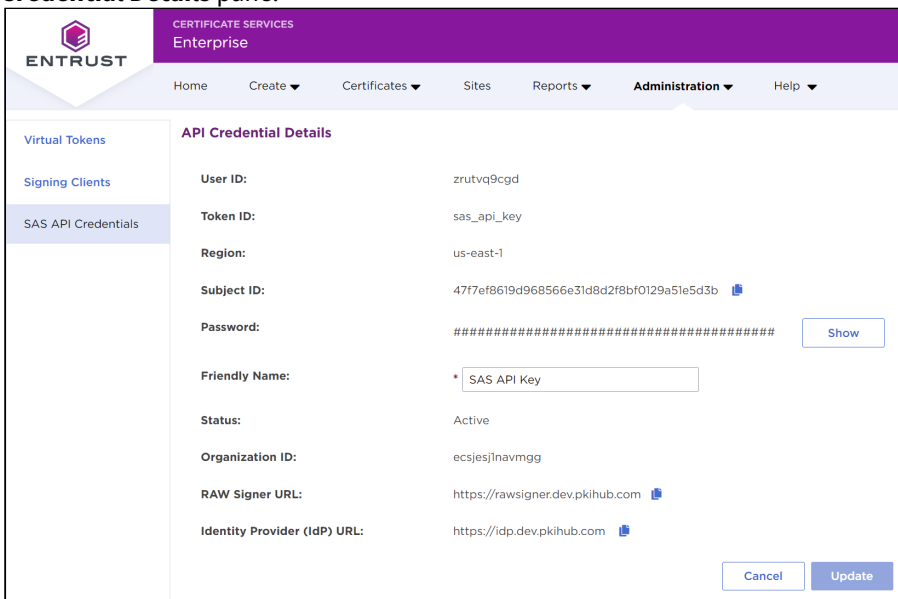
5. In the **Virtual Token ID** field, enter a unique identifier for the new API credential.

6. In the **Friendly Name** field, enter a descriptive name for the new API credential.

7. Click **Next** to display the details of the new API credential.



8. Check the API credentials details and click **Submit** to confirm the API credential creation.
9. Click **OK** in the confirmation dialog and wait a few minutes while the API credential is generated.
10. Back to the list of generated API credentials, click the name of the new API credential to display the **API Credential Details** pane.



11. In the **API Credential Details** pane, copy the following credential values:
  - Subject ID
  - Password
  - Organization ID
  - RAW Signer URL
  - Identity Provider (IdP) URL



## 6 Signing Windows files with the Entrust KSP library

Microsoft CNG (Cryptography API: Next Generation) is a Windows-specific cryptographic API for securing Windows-based applications. This tool uses a hash-based approach when signing requests that do not require the transportation of your files and intellectual property.

See below for integrating the Entrust KSP (Key Storage Provider) library with Microsoft CNG and signing Windows files.

- [Entrust KSP library requirements](#)
- [Downloading the Entrust KSP library](#)
- [Installing the Entrust KSP library](#)
- [Installing and configuring Windows SignTool](#)
- [Signing Windows files with SignTool and the Entrust KSP library](#)

### Entrust KSP library requirements

To sign Windows files using the Entrust KSP library, you need a machine with with the following requirements.

- [Supported Windows versions](#)
- [Supported Microsoft .NET Framework version](#)
- [Network requirements](#)

**i** When [Installing and configuring Windows SignTool](#) you will need to download the Windows Software Development Kit (SDK).

### Supported Windows versions

You need a machine with one of the following Microsoft Windows operating system versions.

- Windows 8.1
- Windows 10
- Windows 11
- Windows Server 2012
- Windows Server 2016
- Windows Server 2019
- Windows Server 2022

### Supported Microsoft .NET Framework version

Install Microsoft .NET Framework 4.8.

### Network requirements

There cannot be a proxy between the machine and the [entrust.net](https://entrust.net) domain.

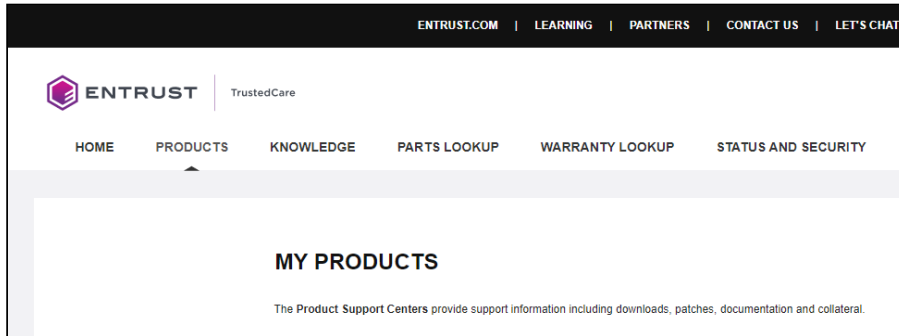
### Downloading the Entrust KSP library

Download the compressed file containing the Signing Automation Client Tools.

#### To download the Signing Automation Client Tools

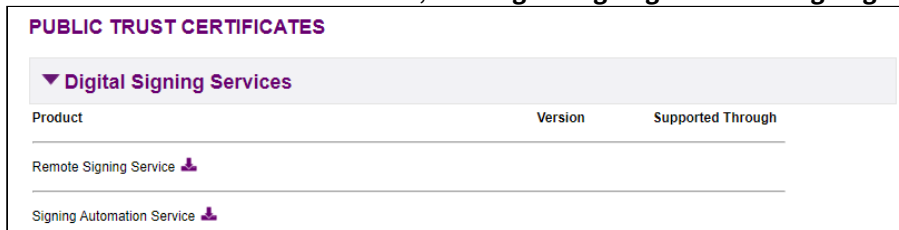
1. Log in to [trustedcare.entrust.com](https://trustedcare.entrust.com)

- Go to **PRODUCTS**.



The screenshot shows the Entrust website navigation bar with links for ENTRUST.COM, LEARNING, PARTNERS, CONTACT US, and LET'S CHAT. Below the navigation bar, the 'MY PRODUCTS' section is displayed, featuring a sub-header and a brief description: 'The Product Support Centers provide support information including downloads, patches, documentation and collateral.'

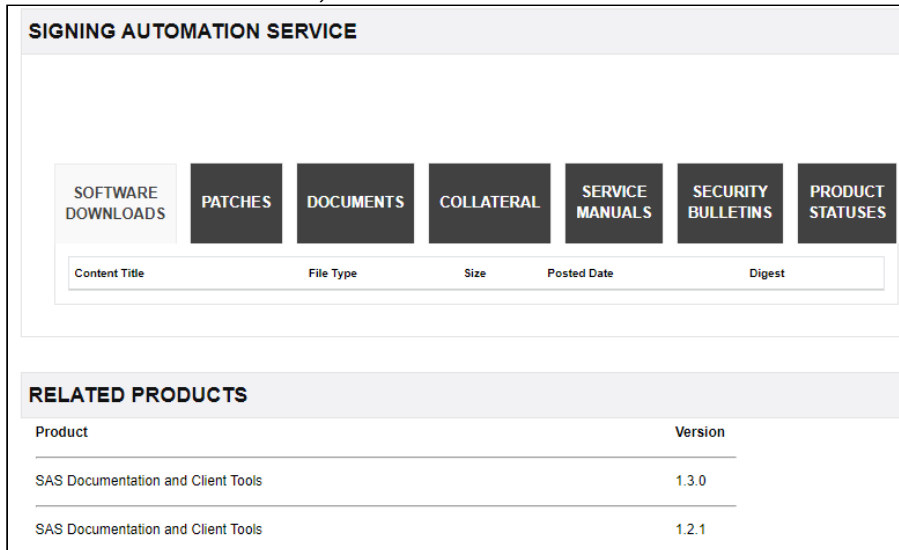
- Under **PUBLIC TRUST CERTIFICATES**, click **Digital Signing Services > Signing Automation Service**.



The screenshot shows the 'PUBLIC TRUST CERTIFICATES' section with a dropdown menu for 'Digital Signing Services'. Below the dropdown, a table lists the services:

Product	Version	Supported Through
Remote Signing Service		
Signing Automation Service		

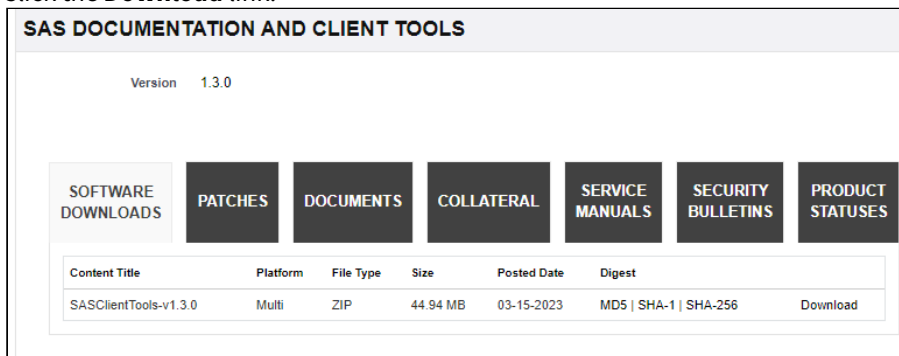
- Under **RELATED PRODUCTS**, click the latest **SAS Documentation and Client Tools** version.



The screenshot shows the 'SIGNING AUTOMATION SERVICE' page with navigation tabs for SOFTWARE DOWNLOADS, PATCHES, DOCUMENTS, COLLATERAL, SERVICE MANUALS, SECURITY BULLETINS, and PRODUCT STATUSES. Below the tabs is a table with columns for Content Title, File Type, Size, Posted Date, and Digest. Further down, the 'RELATED PRODUCTS' section contains a table:

Product	Version
SAS Documentation and Client Tools	1.3.0
SAS Documentation and Client Tools	1.2.1

- Click the **Download** link.

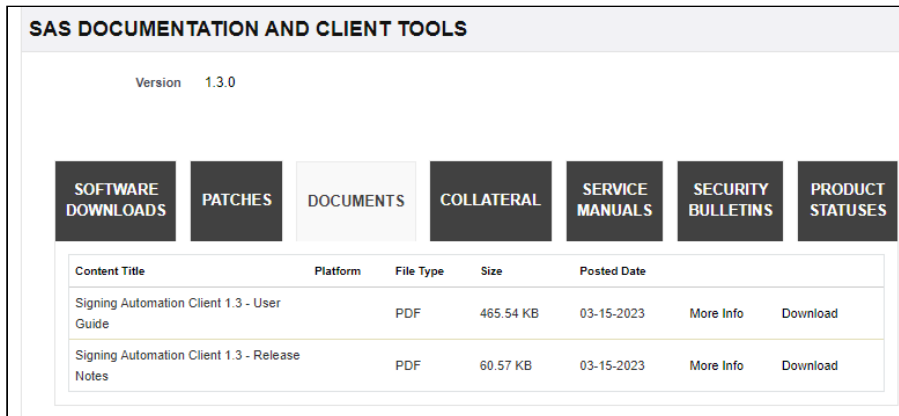


The screenshot shows the 'SAS DOCUMENTATION AND CLIENT TOOLS' page for version 1.3.0. It features navigation tabs and a table with columns for Content Title, Platform, File Type, Size, Posted Date, Digest, and a 'Download' link.

Content Title	Platform	File Type	Size	Posted Date	Digest	
SASClientTools-v1.3.0	Multi	ZIP	44.94 MB	03-15-2023	MD5   SHA-1   SHA-256	Download

- Click **ACCEPT** in the **License Agreement** to download a zipped file containing the Signing Automation Client Client Tools.

- Click the **DOCUMENTS** tab.



- Click the **Download** link for the latest Signing Automation Client documentation.

## Installing the Entrust KSP library

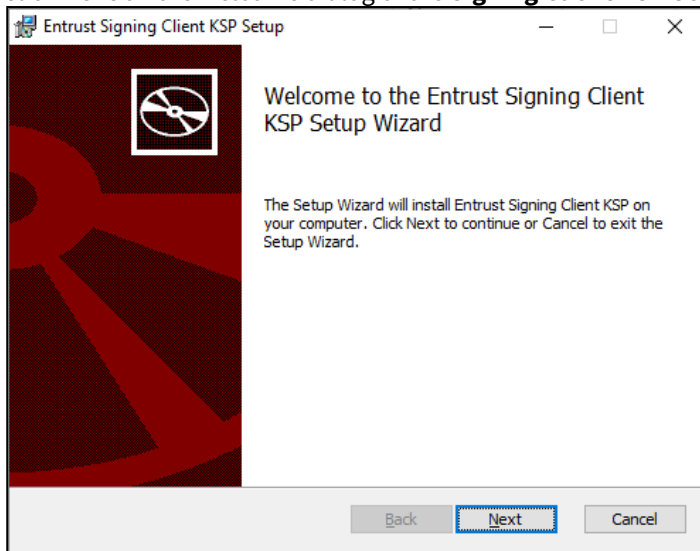
See below for how to install the Entrust KSP library with the Entrust Signing Client KSP Setup wizard.

### To install the Entrust KSP library

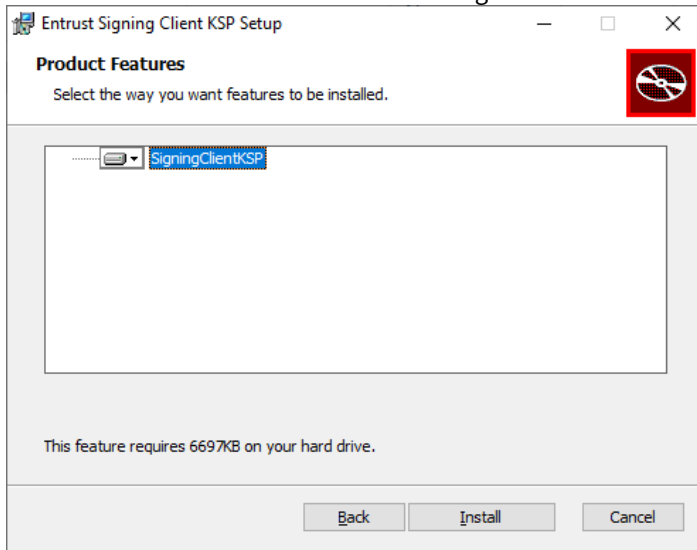
- If not already created, create API credentials as explained in [Getting API credentials](#).
- Download the Signing Automation Client Tools as explained in [Downloading the Entrust KSP library](#).
- Extract the contents of the Signing Automation Client Tools zipped file.
- Run the following executable on a machine with the [Entrust KSP library requirements](#).

```
SASClientTools-v1.3.0\SASClientTools\Software\SigningClientKSP.msi
```

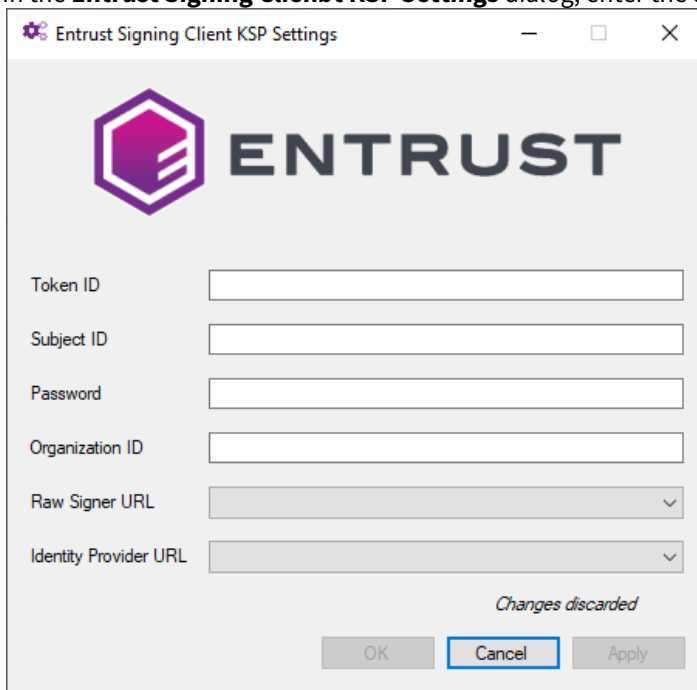
- Click **Next** on the welcome dialog of the **Signing Client KSP Setup** wizard.



6. Click **Install** in the **Product Features** dialog.



7. Wait while the installation completes.
8. In the **Entrust Signing Client KSP Settings** dialog, enter the data obtained when [Getting API credentials](#).



9. Click **OK**.

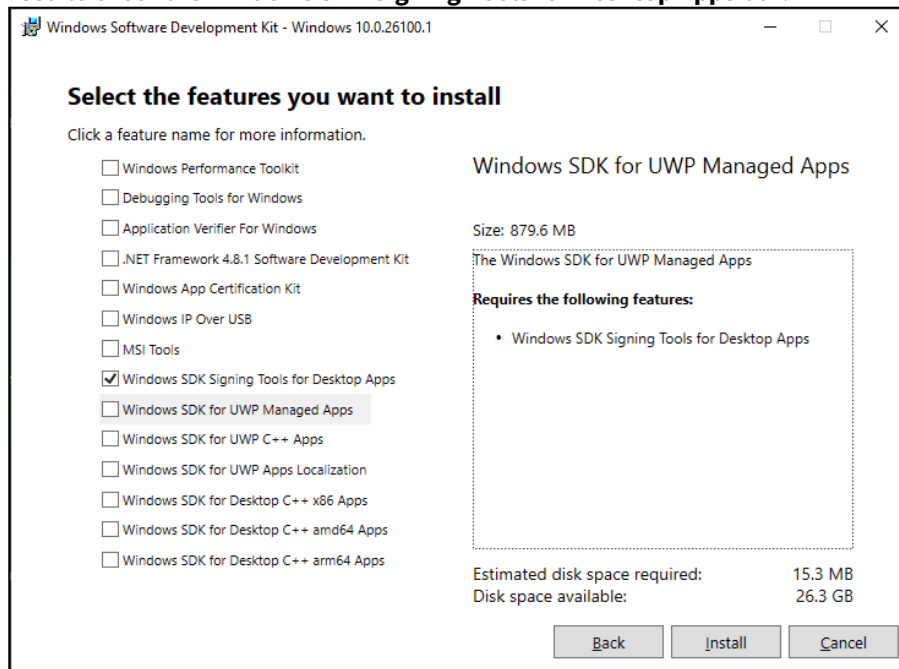
## Installing and configuring Windows SignTool

If not already installed on your Windows machine, install SignTool as explained below.

### To install and configure Windows SignTool

1. Open a web browser in <https://learn.microsoft.com/en-us/windows/win32/seccrypto/signtool>
2. Click the **Windows Software Development Kit (SDK)** link
3. Click **Download the installer**.

- Complete the steps of the installation wizard. In the **Select the features you want install** dialog, you only need to check the **Windows SDK Signing Tools for Desktop Apps** box.



- When completing the installation, edit the **Path** system variable of the Windows machine and add the path of the folder containing the `signtool.exe` file. For example:

```
c:\Program Files (x86)\Windows Kits\10\bin\10.0.26100.0\x64
```

## Signing Windows files with SignTool and the Entrust KSP library

To sign files with SignTool – or other supported tools – and the Entrust KSP library, you need to identify the Entrust Code Signing Certificate to use, either by:

- The certificate thumbprint.
- The **CN** field of the Subject value.

You can obtain both values by running the following command in a PowerShell console.

```
get-childitem cert:\CurrentUser\MY
```

For example:

```
>Get-ChildItem Cert:\CurrentUser\My

    PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My

Thumbprint                               Subject
-----
5226DA79*****17F23832800BA2A          CN=Entrust, .....
```

See below for how to sign Windows files with these values.

- [Running SignTool with the Entrust validation certificate CN](#)
- [Running SignTool with the Entrust validation certificate thumbprint](#)

## Running SignTool with the Entrust validation certificate CN

The following command signs the Windows file in the `<file_path>` path and identifies the Entrust validation certificate with the `<subject_cn>` subject CN.

```
signtool sign /n "<subject_cn>" /fd SHA256 /tr "http://timestamp.entrust.net/
rfc3161ts2" /td SHA256 <file_path>
```

For example:

```
>signtool sign /n "Entrust" /fd SHA256 /tr "http://timestamp.entrust.net/rfc3161ts2"
/td SHA256 ./demo.exe
Done Adding Additional Store
Successfully signed: ./demo.exe
```

## Running SignTool with the Entrust validation certificate thumbprint

The following command signs the Windows file in the `<file_path>` path and identifies the Entrust validation certificate with the `<cert_thumbprint>` thumbprint.

```
signtool sign /sha1 "<cert_thumbprint>" /fd SHA256 /tr "http://timestamp.entrust.net/
rfc3161ts2" /td SHA256 <file_path>
```

For example:

```
>signtool sign /sha1 "5226DA79*****17F23832800BA2A" /fd SHA256 /tr "http://
timestamp.entrust.net/rfc3161ts2" /td SHA256 ./demo.exe
Done Adding Additional Store
Successfully signed: ./demo.exe
```

## 7 Signing with the Entrust PKCS #11 library

Entrust provides a PKCS #11 library to automate signing with third-party applications. See below for how to install and use this library to automate signatures.

- [Installing the PKCS #11 library with the Signing Automation Client](#)
- [Integrating the PKCS #11 library with third-party applications](#)

### Installing the PKCS #11 library with the Signing Automation Client

Install and manage the PKCS #11 library using the Entrust Signing Automation Client.

- [Signing Automation Client requirements](#)
- [Registering a Signing Client in the Entrust Certificate Services](#)
- [Downloading the Signing Automation License](#)
- [Downloading the Signing Automation Client](#)
- [Installing the Signing Automation Client](#)
- [signingclient command-line reference](#)
- [Uninstalling the Signing Automation Client](#)

### Signing Automation Client requirements

The Signing Automation Client supports the following operating systems.

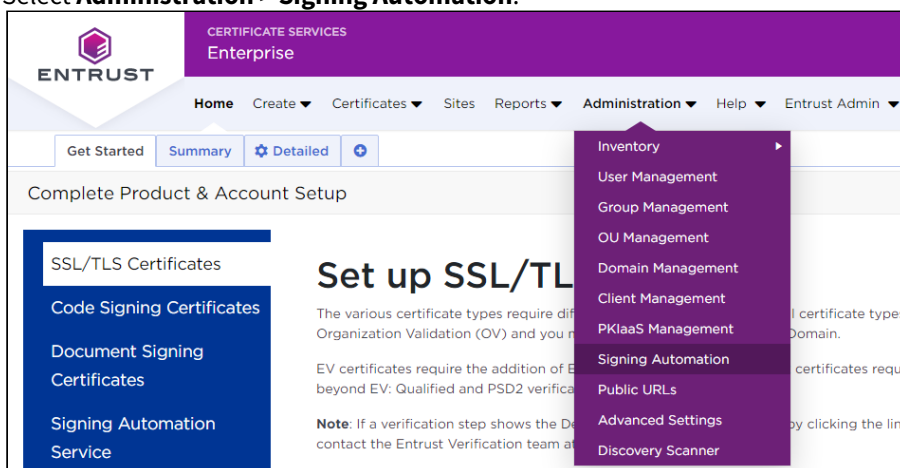
- Windows 10, version 1909 (64-bit)
- Ubuntu 20.04.x (64-bit)

### Registering a Signing Client in the Entrust Certificate Services

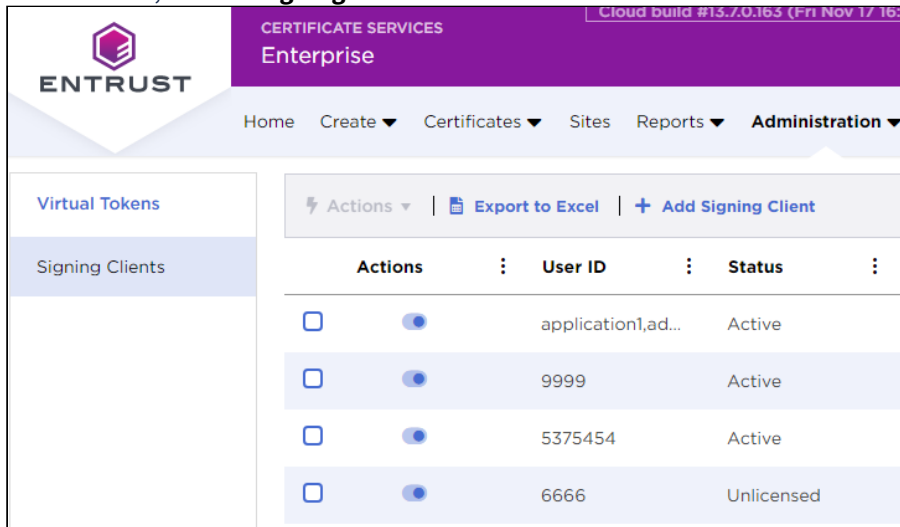
In the Entrust Certificate Services web portal, register a Signing Client with permission on the keys and certificates of a Virtual Token.

#### To register a Signing Client in Entrust Certificate Services

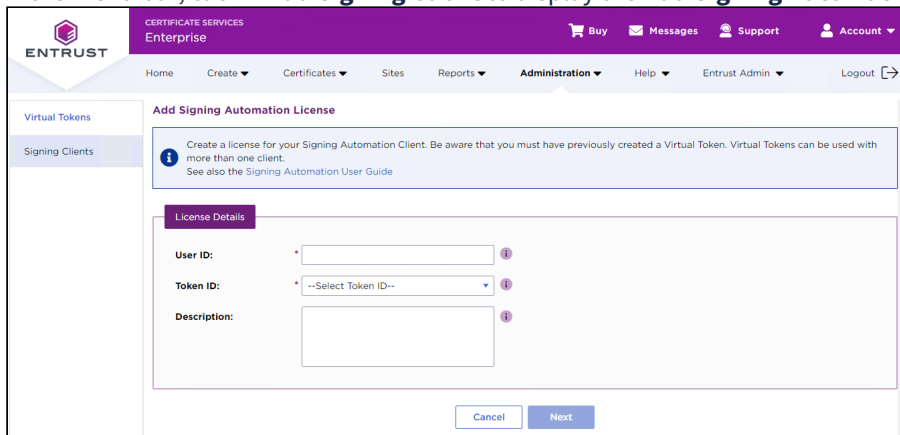
1. Log in to the Entrust Certificate Services web portal at [cloud.entrust.net](https://cloud.entrust.net)
2. Select **Administration > Signing Automation**.



- In the sidebar, click on **Signing Clients**.



- In the menu bar, click **+ Add Signing Client** to display the **Add Signing Automation License** form.



- In the **User ID** field, enter a 3-14 character name to uniquely identify the new Signing Client.
- In the **Token ID** list, select a Virtual Token for the new Signing Client. The Signing Client will be granted permission to sign with the keys and certificates of the selected Virtual Token.
- In the **Description** field, enter an optional description of the new Signing Client.
- Click **Next**.
- Click **Submit** to confirm the Signing Client creation.

## Downloading the Signing Automation License

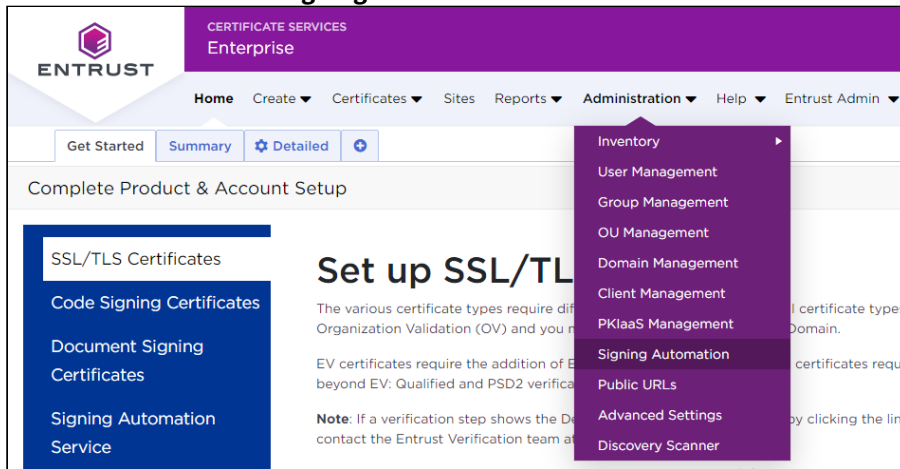
Download your Signing Automation License from the Entrust Certificate Services web portal.

### To download the Signing Automation License

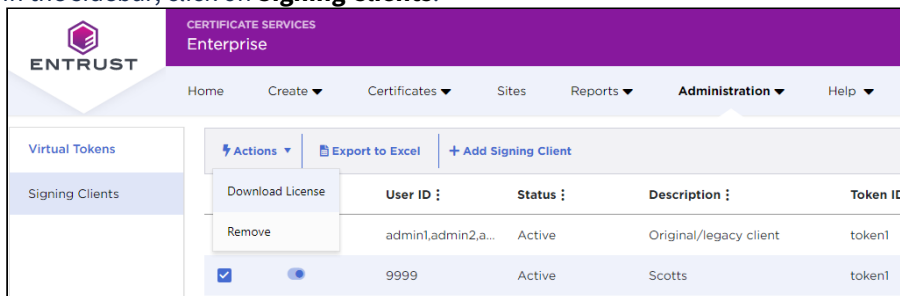
- Log in to the Entrust Certificate Services web portal at [cloud.entrust.net](https://cloud.entrust.net)



2. Select **Administration > Signing Automation**.



3. In the sidebar, click on **Signing Clients**.



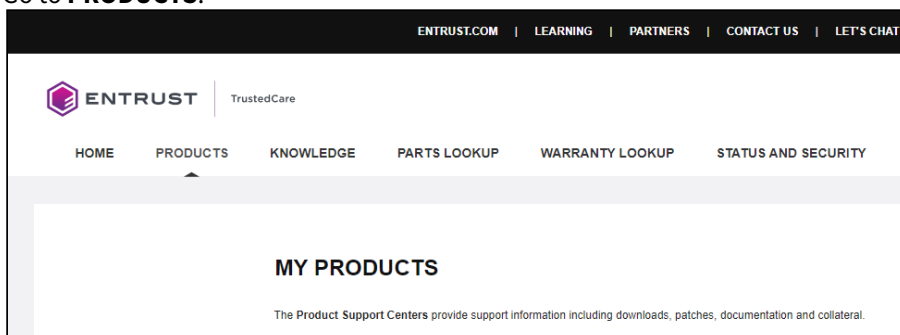
4. In the Signing Clients grid, click the checkbox of a Signing Client row.
5. In the menu bar, select **Actions > Download License**.
6. In the confirmation dialog, click **Download**.

## Downloading the Signing Automation Client

Download the compressed file containing the Signing Automation Client Tools.

### To download the Signing Automation Client Tools

1. Log in to [trustedcare.entrust.com](https://trustedcare.entrust.com)
2. Go to **PRODUCTS**.



3. Under **PUBLIC TRUST CERTIFICATES**, click **Digital Signing Services > Signing Automation Service**.

**PUBLIC TRUST CERTIFICATES**

▼ **Digital Signing Services**

Product	Version	Supported Through
Remote Signing Service		
Signing Automation Service		

4. Under **RELATED PRODUCTS**, click the latest **SAS Documentation and Client Tools** version.

**SIGNING AUTOMATION SERVICE**

SOFTWARE DOWNLOADS | PATCHES | DOCUMENTS | COLLATERAL | SERVICE MANUALS | SECURITY BULLETINS | PRODUCT STATUSES

Content Title	File Type	Size	Posted Date	Digest

**RELATED PRODUCTS**

Product	Version
SAS Documentation and Client Tools	1.3.0
SAS Documentation and Client Tools	1.2.1

5. Click the **Download** link.

**SAS DOCUMENTATION AND CLIENT TOOLS**

Version 1.3.0

SOFTWARE DOWNLOADS | PATCHES | DOCUMENTS | COLLATERAL | SERVICE MANUALS | SECURITY BULLETINS | PRODUCT STATUSES

Content Title	Platform	File Type	Size	Posted Date	Digest	
SASClientTools-v1.3.0	Multi	ZIP	44.94 MB	03-15-2023	MD5   SHA-1   SHA-256	Download

6. Click **ACCEPT** in the **License Agreement** to download a zipped file containing the Signing Automation Client Tools.

7. Click the **DOCUMENTS** tab.

**SAS DOCUMENTATION AND CLIENT TOOLS**

Version 1.3.0

SOFTWARE  
DOWNLOADS

PATCHES

DOCUMENTS

COLLATERAL

SERVICE  
MANUALS

SECURITY  
BULLETINS

PRODUCT  
STATUSES

Content Title	Platform	File Type	Size	Posted Date		
Signing Automation Client 1.3 - User Guide		PDF	465.54 KB	03-15-2023	<a href="#">More Info</a>	<a href="#">Download</a>
Signing Automation Client 1.3 - Release Notes		PDF	60.57 KB	03-15-2023	<a href="#">More Info</a>	<a href="#">Download</a>

8. Click the **Download** link for the latest Signing Automation Client documentation.

## Installing the Signing Automation Client

See below for how to install the Signing Automation Client on Windows and Linux machines.

### To install the Signing Automation Client

1. Download the Signing Automation Client Tools as explained in [Downloading the Signing Automation Client](#).
2. Extract the contents of the Signing Automation Client Tools zipped file.
3. Follow the steps below, depending on your chosen operating system.
  - [Installing the Signing Automation Client in Windows](#)
  - [Installing the Signing Automation Client in Linux](#)

### Installing the Signing Automation Client in Windows

To install the Signing Automation Client in Windows, run the following installer as a user with administrator privileges.

SASDocAndClientTools\Software\SigningClient64.msi

The installer will add the application folder to the `PATH` variable.

C:\Program Files\Entrust\SigningClient

See the following table for the files in this folder.

File	Description
P11SigningClient64.dll	The application library for 64-bit Windows systems.
pkcs11-logger64.dll	The library for <a href="#">Debugging the Signing Automation Client</a> in 64-bit systems.
SigningClient.exe	The application executable.

## Installing the Signing Automation Client in Linux

o install the Signing Automation Client on a Linux platform, extract the contents of the following file to a folder of your choice.

```
SASDocAndClientTools/Software/signingclient64.zip
```

This compressed file contains the following.

File	Description
libp11signingclient64.so	The application library for 64-bit systems.
pkcs11-logger64.so	The library for <a href="#">Debugging the Signing Automation Client in 64-bit systems</a> .
signingclient	The application installer.

If required, add execution permissions to the `signingclient` installer.

```
$ chmod +x signingclient
```


Run the `signingclient` installer.

```
$ ./signingclient
```

## signingclient command-line reference

The `signingclient` executable of the Signing Client supports the commands described below.

- [signingclient completion](#)
- [signingclient config list](#)
- [signingclient config set](#)
- [signingclient create key](#)
- [signingclient credentials](#)
- [signingclient delete certificate](#)
- [signingclient delete key](#)
- [signingclient help](#)
- [signingclient import certificate](#)
- [signingclient list certificates](#)
- [signingclient list keys](#)
- [signingclient process license](#)
- [signingclient version](#)

 See [Installing the Signing Automation Client](#) for the path of the `signingclient` executable.

## signingclient completion

Enables tab completion for `signingclient` commands.

```
signingclient completion <shell> [--log <file>] [--verbose]
```

See below for the supported options.

- `<shell>`
- `--log <file>`
- `--verbose`

`<shell>`

Generate a completion script for the `<shell>` shell. Where `<shell>` is one of the following:

- `bash`
- `fish`
- `powershell`
- `zsh`

Run the following command for instructions on how to install the generated script.

```
signingclient completion <shell> -h
```

For example:

```
$ signingclient completion bash -h
Generate the autocompletion script for the bash shell.

This script depends on the 'bash-completion' package.
If it is not installed already, you can install it via your OS's package manager.

To load completions in your current shell session:

    source <(signingclient completion bash)

To load completions for every new session, execute once:

#### Linux:

    signingclient completion bash > /etc/bash_completion.d/signingclient

#### macOS:

    signingclient completion bash > $(brew --prefix)/etc/bash_completion.d/
signingclient

You will need to start a new shell for this setup to take effect.
```

**Mandatory:** Yes.

--log <file>

Record the command execution in a log file with the <file> path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

--verbose

Print additional error information (if any).

**Mandatory:** No.

### signingclient config list

Lists the user settings.

```
signingclient config list [--log <file>] [--verbose]
```

For example:

```
>signingclient config list
User ID: application1
Subject ID: 0123456789abcdef0123456789abcdef01234567
Organization ID: dsaas999999
Token ID: token1
PKCS #11 library: P11SigningClient64.dll
Signing server: https://rawsigner.dev.pkihub.com
IdP server: https://idp.dev.pkihub.com
Proxy server: <not set>
Proxy auth: <not set>
```

--log <file>

Record the command execution in a log file with the <file> path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

--verbose

Print additional error information (if any).

**Mandatory:** No.

### signingclient config set

Modifies the user settings.

```
signingclient config set [--library <library>] [--log <file>] [--proxy-auth <usr>:<pwd>] [--proxy-host <host>:<port>] [--token] [--verbose]
```

For example, to set the path of the PKCS #11 library:


```
signingclient config set --library "C:\Program Files\Entrust\SigningClient\P11SigningClient64.dll"
```

To delete the proxy configuration:

```
signingclient config set --proxy-host "" --proxy-auth ""
```

See below for the supported options.

- `--library <library>`
- `--log <file>`
- `--proxy-auth <usr>:<pwd>`
- `--proxy-host <host>:<port>`
- `--token`
- `--verbose`

 The Signing Automation Client and the PKCS #11 library do not support Man-in-the-Middle (MITM) proxy types. Please contact customer support if you want to prioritize MITM proxy support.

`--library <library>`

Use the PKCS #11 library file in the `<library>` path.

**Mandatory:** No.

`--log <file>`

Record the command execution in a log file with the `<file>` path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

`--proxy-auth <usr>:<pwd>`

Authenticate in the proxy as the `<usr>` user with the `<pwd>` password.

**Mandatory:** No.

`--proxy-host <host>:<port>`

Select the `<host>:<port>` proxy.

**Mandatory:** No.

`--token`

List the available tokens for the user to select one.


**Mandatory:** No.

--verbose

Print additional error information (if any).

**Mandatory:** No.

## signingclient create key


 This command is for administrator users only. In a normal scenario, Entrust Certificate Services automatically manages your keys and certificates.

Generates a key pair and the corresponding CSR (Certificate Signing Request).

```
signingclient create key --key-type <key_type> [--csr-out <csr>] [--csr-subject <subject>] [--key-id <id>] [--key-label <label>] [--log <file>] [--password <pwd>] [--verbose]
```

See below for the supported options.

- --csr-out <csr>
- --csr-subject <subject>
- --key-id <id>
- --key-label <label>
- --key-type <key\_type>
- --log <file>
- --password <pwd>
- --verbose

 The command signs the CSR and, therefore, consumes one of the 10,000 licensed signatures.


--csr-out <csr>

Save the generated CSR in the <csr> file path.

**Mandatory:** No. When omitting this option, the command skips the CSR generation.

--csr-subject <subject>

Use <subject> as the Subject of the certificate request. Where <subject> is a full Distinguished Name (DN) or Relative Distinguished Name (RDN).

 For Entrust Validation Authority to recognize the Subject, the DN attributes must be in capital letters.

For example:

```
CN=Example User,O=Example,C=US
```



```
CN=Example User
```

**Mandatory:** No. When omitting this option, the Subject in the generated certificate request defaults to the following:

```
CN=<key_id>
```

Where `<key_id>` is the key identifier.

```
--key-id <id>
```

Set `<id>` as the hexadecimal key identifier.

**Mandatory:** No. When omitting this option, the identifier is the public key's SHA1.

```
--key-label <label>
```

Set `<label>` as the key label.

**Mandatory:** No. When omitting this option, the label is the key identifier.

```
--key-type <key_type>
```

Create a key of the `<key_type>` type, where `<key_type>` is one of the following.

- RSA2048
- RSA3072
- RSA4096
- ECDSAP256
- ECDSAP384
- ECDSAP521

**Mandatory:** Yes.

```
--log <file>
```

Record the command execution in a log file with the `<file>` path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

```
--password <pwd>
```

Set `<pwd>` as the token password.

**Mandatory:** No. When omitting this option, the command prompts for the password value.

```
--verbose
```

Print additional error information (if any).

**Mandatory:** No.

## signingclient credentials

Manages the user credentials

```
signingclient credentials --change-password [--log <file>] [--verbose]
```

See below for the supported options.

- `--change-password`
- `--log <file>`
- `--verbose`

`--change-password`

Change the user password – for example:

```
>signingclient credentials --change-password
Password:
Enter the new password to protect your credentials
New password:
Confirm password:
Writing credentials to: /home/user/.signingclient/credentials
Password changed
```

**Mandatory:** Yes. This is currently the only supported credential management option.

`--log <file>`

Record the command execution in a log file with the `<file>` path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.


**Mandatory:** No. When omitting this option, the command does not record a log.

`--verbose`

Print additional error information (if any).

**Mandatory:** No.

## signingclient delete certificate

 This command is for administrator users only. In a normal scenario, Entrust Certificate Services automatically manages your keys and certificates.

Deletes a certificate.

```
signingclient delete certificate --cert-id <cert_id> [--force] [--log <file>] [--verbose]
```

See below for the supported options.

- `--cert-id <cert_id>`

- `--force`
- `--log <file>`
- `--verbose`

⚠ Deleting the certificates obtained when [Creating a document signing certificate](#) makes the Signing Automation Service unusable.

`--cert-id <cert_id>`

Delete the certificate with the `<id>` identifier.

**Mandatory:** Yes.

`--force`

Skip the confirmation before deleting the certificate.

**Mandatory:** No. When omitting this option, the command prompts for confirmation.

`--log <file>`

Record the command execution in a log file with the `<file>` path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

`--verbose`

Print additional error information (if any).

**Mandatory:** No.

## signingclient delete key

⚠ This command is for administrator users only. In a normal scenario, Entrust Certificate Services automatically manages your keys and certificates.

Deletes a key.

```
signingclient delete key --key-id <key_id> [--force] [--log <file>] [--verbose]
```

See below for the supported options.

- `--force`
- `--key-id <key_id>`
- `--log <file>`
- `--verbose`

⚠ Deleting the key obtained in [Creating a document signing certificate](#) makes the Signing Automation Service unusable.

--force

Skip the confirmation before deleting the key.

**Mandatory:** No. When omitting this option, the command prompts for confirmation.

--key-id <key\_id>

Delete the key with the <id> identifier.

**Mandatory:** Yes.

--log <file>

Record the command execution in a log file with the <file> path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

--verbose

Print additional error information (if any).

**Mandatory:** No.

## signingclient help

Prints help information on `signingclient` subcommands. You can use any of the following syntaxes.

```
signingclient help <subcommands>
```

```
signingclient <subcommands> -h
```

```
signingclient <subcommands> --help
```


Where <subcommands> is the list of subcommands. For example, the following commands display the same help information on the `signingclient list certificates` command.

```
signingclient help list certificates
```

```
signingclient list certificates -h
```

```
signingclient list certificates --help
```

## signingclient import certificate

 This command is for administrator users only. In a normal scenario, Entrust Certificate Services automatically manages your keys and certificates.

Imports a certificate.

```
signingclient import certificate <cert_file> [--cert-id <id>] [--cert-label <label>]
[--no-trusted] [--log <file>] [--verbose]
```

See below for the supported options.

- <cert\_file>
- --cert-id <id>
- --cert-label <label>
- --log <file>
- --no-trusted
- --verbose

<cert\_file>

Import the certificate in the <cert\_file> file path.

**Mandatory:** Yes.

--cert-id <id>


Set <id> as the hexadecimal identifier of the certificate.

**Mandatory:** No. When omitting this option, the identifier is:

- The public key identifier, if the certificate public key is in the token.
- The public key SHA1 otherwise.

--cert-label <label>

Set <label> as the certificate label.

 When [Integrating the PKCS #11 library with third-party applications](#), the keystore of the third-party application must use the same <label> label to identify the certificate.

**Mandatory:** No. When omitting this option, the certificate label is the certificate subject.

--log <file>

Record the command execution in a log file with the <file> path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

--no-trusted

Set CKA\_TRUSTED to false in the certificate flags.

! Setting `CKA_TRUSTED` to `false` can prevent Java signing applications from working.

**Mandatory:** No. When omitting this option, `CKA_TRUSTED` is `true`.

--verbose

Print additional error information (if any).

**Mandatory:** No.

### signingclient list certificates

List the token certificates.

```
signingclient list certificates [--log <file>] [--verbose]
```

--log <file>

Record the command execution in a log file with the `<file>` path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

--verbose

Print additional error information (if any).

**Mandatory:** No.

### signingclient list keys

Lists the token keys.

```
signingclient list keys [--log <file>] [--verbose]
```

--log <file>

Record the command execution in a log file with the `<file>` path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

--verbose

Print additional error information (if any).

**Mandatory:** No.

## signingclient process license

Imports a license key.

```
signingclient process license <license> [--all-available-users] [--api-credentials]
[--force] [--log <file>] [--verbose]
```

See below for the supported options.

- <license>
- --all-available-users
- --api-credentials
- --force
- --log <file>
- --verbose


<license>

Process the <license> license, where <license> is the path license file. See [Downloading the Signing Automation License](#) for how to obtain this file.

**Mandatory:** Yes.

--all-available-users

Print the list of licensed users.


 This command is maintained for backward compatibility with legacy multi-user licenses.

**Mandatory:** No.

--api-credentials

Print credentials for the REST API instead of generating the [config](#) and [credentials](#) files described in [Installing the Signing Automation License](#). For example:

```
>signingclient process license intsasapi1.lic --api-credentials
Organization ID: intsasapi1
Choose the user to configure:
  1) admin1
Select number [1-1]: 1
User ID: admin1
WARNING: If you continue the license will be consumed and your credentials to access
the API will be printed on the screen.
      This will be the only time they will be displayed, so make sure you save
them securely.
NOTE: The API credentials will NOT be compatible with the PKCS #11 library
Continue? (y/n): y
{
  "subjectID": "a08ba7436ecc63cdc09254c6768c2ddefac4e57b",
  "secret": "1570b864d25c5102918066ccb2b470501b89872c"
}
Process license OK
```

 The printed credentials are not valid for running the operations provided by the PKCS#11 library.

**Mandatory:** No. When omitting this flag, the command saves the [config](#) and [credentials](#) files described in [Installing the Signing Automation License](#).

--force

Overwrite the [config](#) and [credentials](#) files described in [Installing the license key](#).

 As explained in [Setting the P11PKIHUB\\_CONFIG variable](#), you can select the path of these files.

**Mandatory:** No. When omitting this flag, the application raises an exception if existing user files already exist.

--log <file>

Record the command execution in a log file with the `<file>` path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

--verbose

Print additional error information (if any).

**Mandatory:** No.

signingclient version

Prints the version of the Entrust Automation Signing Client.

```
signingclient version [--log <file>] [--verbose]
```



For example:

```
>signingclient version
CLI Version: v202103090152
Library: C:\Program Files\Entrust\SigningClient\P11SigningClient64.dll
Library Version: 1.0
Library Description: Signing Client v202102181732
```

`--log <file>`

Record the command execution in a log file with the `<file>` path.

- If the file does not exist, the command creates it.
- If the file exists, the command appends the execution log.

**Mandatory:** No. When omitting this option, the command does not record a log.

`--verbose`

Print additional error information (if any).

**Mandatory:** No.

## Uninstalling the Signing Automation Client

To uninstall the Signing Automation Client in Windows, you can:

- Run the installer, and select **Remove**.
- Use the Windows **Add or remove programs** dialog

To uninstall the Signing Automation Client in Linux, remove the installation files.

## Integrating the PKCS #11 library with third-party applications

After [Installing the PKCS #11 library with the Signing Automation Client](#), you can integrate this library with third-party signing applications.

- [Requirements for integrating third-party signing applications](#)
- [Signing PDF documents with iText](#)
- [Signing Microsoft Authenticode files with Jsign](#)
- [Signing Java files with Jarsigner](#)

## Requirements for integrating third-party signing applications

Before integrating a third-party signing application, you must perform the following steps.

- [Setting the P11PKIHUB\\_CONFIG variable](#)
- [Installing the Signing Automation License](#)

### Setting the P11PKIHUB\_CONFIG variable

If required, set the `P11PKIHUB_CONFIG` environment variable to modify the location for the user settings.

OS	Default user settings location
Linux	/home/\$USERNAME/.signingclient
Windows	C:\Users\%USERNAME%\AppData\Roaming\Entrust\SigningClient

## Installing the Signing Automation License

Run the `signingclient process license` command to install the following types of Signing Automation License.

- Signed files with the `.lic` extension. See [Downloading the Signing Automation License](#) for how to obtain these files.
- Password-protected zip files. The signing client supports this legacy format for backward compatibility.

For example:

```
>SigningClient.exe process license intsasapi2w.lic
Organization ID: intsasapi2w
Choose the user to configure:
  1) admin1
Select number [1-1]: 1
User ID: admin1
Create a password to protect your credentials
New password:
Confirm password:
Writing credentials to: C:
\Users\romeror\AppData\Roaming\Entrust\SigningClient\credentials
Writing config to: C:\Users\romeror\AppData\Roaming\Entrust\SigningClient\config
Choose the token to configure:
  1) token82809
Select number [1-1]: 1
Token ID: token82809
Writing config to: C:\Users\romeror\AppData\Roaming\Entrust\SigningClient\config
Process license OK
```

See below for a description of the generated files.

- [config](#)
- [credentials](#)

 See [Setting the P11PKIHUB\\_CONFIG variable](#) for how to change the default path of these files.

`config`

The command-line tool saves the user settings in the following file when completing the license key installation.

OS	Default path
Windows	C:\Users\%USERNAME%\AppData\Roaming\Entrust\SigningClient\config
Linux	\$HOME/.signingclient/config

As explained in the [signingclient command-line reference](#), you can manage these configuration settings with the following commands:

- [signingclient config list](#)
- [signingclient config set](#)

For example, to update the library path:

```
$ signingclient config set --library /home/john/your_library_path/libp11pkihub.so
```

credentials

With the password entered by the administrator, the command-line tool encrypts the user secret in the following file.


OS	Default path
Windows	C:\Users\%USERNAME%\AppData\Roaming\Entrust\SigningClient\credentials
Linux	\$HOME/.signingclient/credentials

 We strongly recommend backing up this file.

## Signing PDF documents with iText

Once installed and configured, you can integrate the Signing Automation Client with a signing application. For example, you can run the signature samples provided with the iText library for Java.

### To run the iText signature samples

1. Configure a service subscription as explained in [Managing tokens with Entrust Certificate Services cloud](#).
2. On your computer, install:
  - The PKCS #11 library, as explained in [Installing the PKCS #11 library with the Signing Automation Client](#).
  - Oracle JDK 8
  - The Eclipse IDE
3. Open a web browser in <https://github.com/itext/i7js-signatures> 
4. Click on **Code > Download ZIP**
5. Download the ZIP file on your computer.
6. Unzip the downloaded file into a working folder on your PC. For example:

```
C:\Projects\i7js-signatures-develop
```

7. Launch the Eclipse IDE.
8. Select **File > Import > General > Projects from Folder or Archive**.
9. In **Import source**, indicate the path of the unzipped iText samples.
10. Click **Finish**.
11. Add a subfolder to the project. For example:

```
C:\Projects\i7js-signatures-develop\config
```

12. In this new subfolder, create the files described in the following sections.
  - [hsm.properties](#)
  - [signkey.properties](#)
13. In the Eclipse navigation pane, browse to **i7js-signatures-develop > src/test/java > com.itextpdf.samples.signatures.chapter04**
14. In the navigation pane, double-click **C4\_01\_SignWithPKCS11HSM.java**
15. Update the file contents as explained in [C4\\_01\\_SignWithPKCS11HSM.java](#).
16. In the Eclipse navigation pane, right-click **C4\_01\_SignWithPKCS11HSM.java** and select **Run As > Java Application**.
17. Open the generated file with Adobe Acrobat Reader.

```
C:\Projects\i7js-signatures-develop\target\signatures\chapter04\hello_hsm.pdf
```

18. Check that the generated PDF file includes an embedded signature and a timestamp.

## hsm.properties

Create an `hsm.properties` file with the following contents.

```
name = Entrust
library = C:\Program Files\Entrust\SigningClient\P11SigningClient64.dll
slot = 1
```

Where `library` is the path of the Signing Automation Client library. For additional variables, see:

<https://docs.oracle.com/javase/8/docs/technotes/guides/security/p11guide.html> 

## signkey.properties

Create a `signkey.properties` file with the following contents.

```
PASSWORD = <password>
PKCS11CFG = ./config/hsm.properties
```

Where:

- `PASSWORD` is the password of the logical token.

- PKCS11CFG is the path of the [hsm.properties](#) file.

## C4\_01\_SignWithPKCS11HSM.java

Update the code of the `C4_01_SignWithPKCS11HSM.java` signature sample as explained in the following sections.

- [Updating the properties path](#)
- [Selecting the certificate](#)
- [Checking certificate selection](#)

### Updating the properties path

The `C4_01_SignWithPKCS11HSM.java` code loads the `signkey.properties` file from the `C:/` root folder.

```
properties.load(new FileInputStream("C:/signkey.properties"));
```

Update this line to use the relative path of the `signkey.properties` file.

```
properties.load(new FileInputStream("../config/signkey.properties"));
```

### Selecting the certificate

The following line of the `C4_01_SignWithPKCS11HSM.java` code selects the alias of the signing certificate.

```
String alias = ks.aliases().nextElement();
```

However, this code line may select the alias of an invalid certificate – for example, when the token includes successive renewals of the signing certificate. To update this code line, list all the certificate labels with the `signingclient list certificates` command.

If each certificate has a unique label, set this line as follows.

```
String alias = "<label>";
```

If different certificates share the same label, set this line as follows.

```
String alias = "<label>/<issuer>/<sn>"
```

Where:

- `<label>` is the label returned by the `signingclient list certificates` command.
- `<issuer>` is the DN (Distinguished Name) of the issuer's certificate.
- `<sn>` is the Serial Number of the certificate in decimal representation, as opposed to the hexadecimal representation printed by the `signingclient list certificates` command.

As explained in the [Oracle documentation](#):

*If multiple certificates share the same CKA\_LABEL, then the alias is derived from the CKA\_LABEL plus the end entity certificate issuer and serial number ("MyCert/CN=foobar/1234", for example).*

## Checking certificate selection

Under this line of the `C4_01_SignWithPKCS11HSM.java` code.

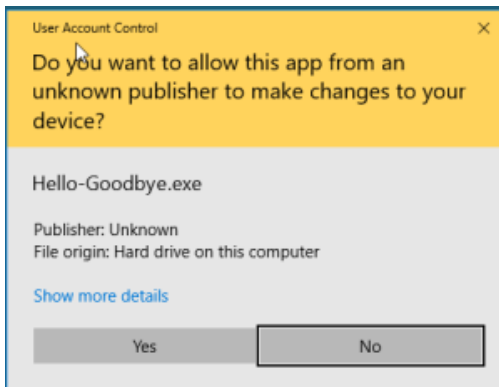
```
Certificate[] chain = ks.getCertificateChain(alias);
```

Add a clause to throw an exception when the selected alias does not correspond to a valid signing certificate.

```
if (pk == null || chain == null) {  
    throw new IllegalArgumentException("Couldn't find a certificate with the  
specified label");  
}
```

## Signing Microsoft Authenticode files with Jsign

Windows SmartScreen displays a warning like the following when an application executable is not signed.



As explained in [ebourg.github.io/jsign](https://ebourg.github.io/jsign), you can skip this warning by signing Windows executables with Jsign, a Java-based implementation of Microsoft Authenticode.

### To sign Windows executables with Jsign

1. Configure a service subscription as explained in [Managing tokens with Entrust Certificate Services cloud](#).
2. Install the PKCS #11 library as explained in [Installing the PKCS #11 library with the Signing Automation Client](#).
3. Follow the steps described below.
  - [Installing and configuring Jsign](#)
  - [Getting the token label](#)
  - [Testing the signing settings](#)
  - [Signing a Windows executable](#)
  - [Verifying the signature](#)

### Installing and configuring Jsign

Install and configure Jsign on your machine.

- [Installing and configuring Jsign in Linux](#)
- [Installing and configuring Jsign on Windows](#)

## Installing and configuring Jsign in Linux

When using a Linux machine, install and configure Jsign as follows.

- [Installing Jsign on Linux](#)
- [Configuring Jsign on Linux](#)

### Installing Jsign on Linux

Run the following commands to install Jsign on a Linux machine.

```
curl -fSsLL <downloadUrl>
sudo dnf localinstall <rpm> -y
```

Where

- `<downloadUrl>` is the download link of the RPM install file for the latest release available at <https://github.com/ebourg/jsign/releases>
- `<rpm>` is the file name of the RPM file.

For example:

```
$ curl -fSsLL https://github.com/ebourg/jsign/releases/download/6.0/
jsign-6.0-1.noarch.rpm
$ sudo dnf localinstall jsign-6.0-1.noarch.rpm -y
```

### Configuring Jsign on Linux

Create a configuration file – for example:

```
name = Entrust-CSaaS-PKCS11
description = Entrust CSaaS PKCS11 Driver
library = /home/eve/libp11signingclient64.so
slotListIndex = 0
```

Where the `library` is the path of the signing library described in [Installing the Signing Automation Client](#).

 The examples below assume `csaas.cfg` as the name of this configuration file.

## Installing and configuring Jsign on Windows

When using a Windows machine, install and configure Jsign as follows.

- [Installing Jsign on Windows](#)
- [Adding Jsign to the Path environment variable in Windows](#)

### Installing Jsign on Windows

To install Jsign on a Windows machine, open a web browser and navigate to the following URL:

<https://github.com/ebourg/jsign/releases>

Download the JAR file of the latest release – for example:

```
jsign-6.0.jar
```

Move the file to a folder of your choice.

Adding Jsign to the Path environment variable in Windows

Add the downloaded JAR file to the **Path** environment variable.


1. Press the Windows key.
2. Type **Edit the system environment variables** or **Edit environment variables for your account** depending on your user permissions.
3. Edit the **Path** environment variable.
4. Add the path of the folder containing the JAR file.

## Getting the token label

Run the `signingclient list keys` command to get the label of your code signing token.

```
$ signingclient list keys
Using token with label csaasdemo
Password:
Private Key Object; RSA 4096 bits
  Label:      csaasdemo
  ID:        876057e0fe3b0d53c822c312f4a7c76a76dd5644
  Usage:     sign

Public Key Object; RSA 4096 bits
  Label:      csaasdemo
  ID:        876057e0fe3b0d53c822c312f4a7c76a76dd5644
  Usage:     encrypt, verify, wrap
```

 Your token might have more than one code signing key. Please make sure you select the right key.

## Testing the signing settings

Run the `keytool -list` command to test:

- The configuration file.
- The access to the Virtual Token in Code Signing as a Service.

For example:

```
$ keytool -list -v -keystore NONE -storetype PKCS11 -providerClass
sun.security.pkcs11.SunPKCS11 -providerArg csaas.cfg
Enter keystore password:

Keystore type: PKCS11
Keystore provider: SunPKCS11-Entrust-CSaaS-PKCS11
Your keystore contains 1 entry
Alias name: csaasdemo
```



```
Entry type: PrivateKeyEntry
Certificate chain length: 3
Certificate[1]:
Owner: CN=Entrust Limited, SERIALNUMBER=1000492879, OID.2.5.4.15=Private
Organization, O=Entrust Limited, OID.1.3.6.1.4.1.311.60.2.1.2=Ontario,
OID.1.3.6.1.4.1.311.60.2.1.3=CA, L=Ottawa, ST=Ontario, C=CA
Issuer: CN=Entrust Extended Validation Code Signing CA - EVCS2, O="Entrust, Inc.",
C=US
Serial number: 60f0b0*****2c457fadb
```

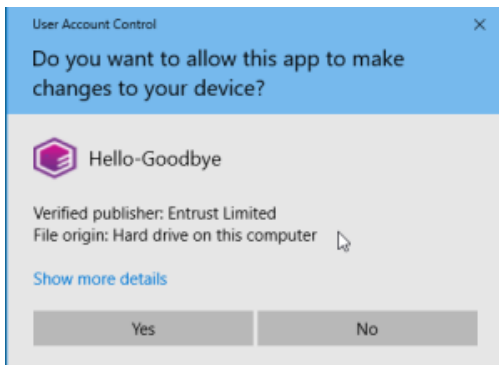
## Signing a Windows executable

Run `jsign` to sign a Windows executable file – for example:

```
$ jsign --keystore csaas.cfg --alias csaasdemo -storepass xxxxxxxxxx --storetype
PKCS11 --tsaurl http://timestamp.entrust.net/rfc3161ts2 --tsmode RFC3161 Hello-
Goodbye.exe
Adding Authenticode signature to Hello-Goodbye.exe
```

## Verifying the signature

When running the signed executable on a Windows machine, the confirmation dialog will display information on the file signer.

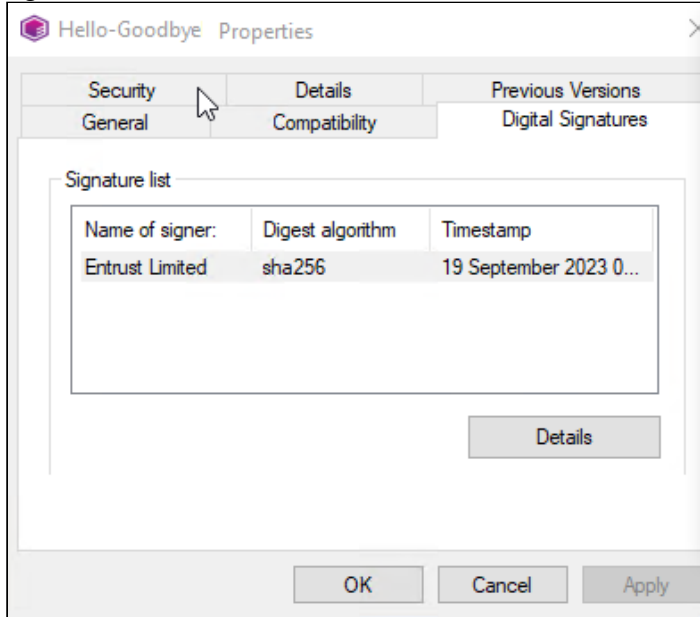


See below how to browse additional information

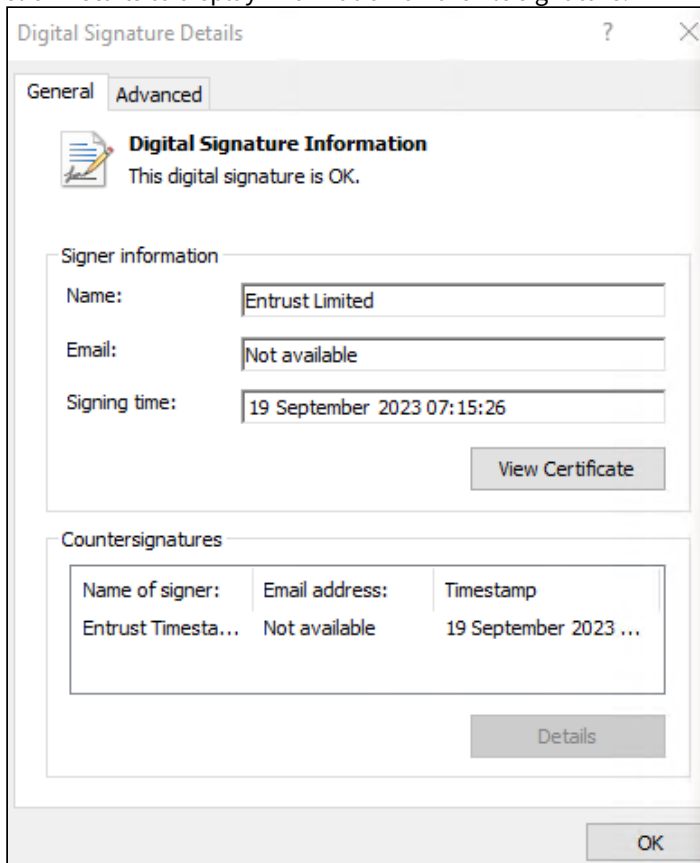
### To browse information on a file signature

1. Locate the file using Windows File Explorer.

- Right-click the file name and select the **Properties** contextual command to display the **Properties** dialog.



- Click **Details** to display information on the file signature.



- Click **View Certificate** to browse the signing certificate details.

## Signing Java files with Jarsigner

Jarsigner is a tool provided with the Java Development Kit (JDK) to digitally sign Java files with the following extensions.

- .ear
- .jar
- .sar
- .war

See below for how to sign these files with Jarsigner and the keys managed by the Entrust Signing Automation service.

### To sign JAVA files with Jarsigner

1. Configure a service subscription as explained in [Managing tokens with Entrust Certificate Services cloud](#).
2. On your computer, install:
  - The Signing Automation Client, as explained in [Downloading the Signing Automation Client](#).
  - The latest Java Development Kit (JDK) available at <https://www.oracle.com/java/technologies/downloads>
3. Follow the steps described below.
  - [Configuring Jarsigner](#)
  - [Running Jarsigner](#)

## Configuring Jarsigner

Depending on the system running your machine, configure Jarsigner as explained in one of the following sections.

- [Configuring Jarsigner in Windows](#)
- [Configuring Jarsigner in Linux](#)

### Configuring Jarsigner in Windows

When using a Windows machine to run Jarsigner, paste the following contents in a text editor.

```
name=<token>
library="<P11SigningClient64.dll>"
```

Where:

- <token> is the `Token ID` value returned by the `signingclient config list` command.
- <P11SigningClient64.dll> is the path of the `P11SigningClient64.dll` library.

For example:

```
name=csaas
library="C:\\Program Files\\Entrust\\SigningClient\\P11SigningClient64.dll"
```

Save these contents as the following file.

```
C:\\Program Files\\Java\\jdk-17\\bin\\csaas.cfg
```

## Configuring Jarsigner in Linux

When using a Linux machine to run Jarsigner, paste the following contents in a text editor.

```
name=<token>
library="<P11SigningClient64.so>"
```

Where:

- `<token>` is the `Token ID` value returned by the `signingclient config list` command.
- `<P11SigningClient64.so>` is the path of the `P11SigningClient64.so` library.

For example:

```
name=csaas
library="/home/admin/signingclient64/pkcs11-logger64.so"
```

Save these contents as the following file.

```
<jdk>/bin/csaas.cfg
```

Where `<jdk>` is the path of the Oracle Java JDK installation – for example:

```
/usr/lib/jvm/jdk-22-oracle-x64/bin/csaas.cfg
```

## Running Jarsigner

Run the following command to generate the timestamped signature of a Java file with Jarsigner.

```
jarsigner -tsa http://timestamp.entrust.net/rfc3161ts2 -verbose -keystore NONE
-storetype PKCS11 -providerClass sun.security.pkcs11.SunPKCS11 -providerArg "C:
\Program Files\Java\jdk-17\bin\csaas.cfg" <jar> <token> -storepass <storepass>
```

Where:

- `<jar>` is the path of the Java file
- `<token>` is the `Token ID` value returned by the `signingclient config list` command.
- `<storepass>` is the password for accessing the signing key.

For example:

```
>jarsigner -tsa http://timestamp.entrust.net/rfc3161ts2 -verbose -keystore NONE
-storetype PKCS11 -providerClass sun.security.pkcs11.SunPKCS11 -providerArg "C:
\Program Files\Java\jdk-17\bin\csaas.cfg" Main.jar myuser -storepass mypass

requesting a signature timestamp

TSA location: http://timestamp.entrust.net/rfc3161ts2
```

```
updating: META-INF/MANIFEST.MF

adding: META-INF/CSAASDEM.SF

adding: META-INF/CSAASDEM.RSA

signing: HelloWorld/Main.class

>>> Signer

X.509, CN=Entrust Limited, SERIALNUMBER=1000492879, OID.2.5.4.15=Private
Organization, O=Entrust Limited, OID.1.3.6.1.4.1.311.60.2.1.2=Ontario,
OID.1.3.6.1.4.1.311.60.2.1.3=CA, L=Ottawa, ST=Ontario, C=CA

Signature algorithm: SHA256withRSA, 4096-bit key

[certificate is valid from 8/16/23, 1:07 PM to 8/16/24, 1:07 PM]

X.509, CN=Entrust Extended Validation Code Signing CA - EVCS2, O="Entrust, Inc.",
C=US

Signature algorithm: SHA512withRSA, 4096-bit key

[certificate is valid from 5/7/21, 7:19 PM to 12/29/40, 11:59 PM]

X.509, CN=Entrust Code Signing Root Certification Authority - CSBR1, O="Entrust,
Inc.", C=US

Signature algorithm: SHA256withRSA, 4096-bit key

[certificate is valid from 5/7/21, 3:43 PM to 11/7/30, 4:13 PM]

>>> TSA

X.509, CN=Entrust Timestamp Authority - TSA2, O="Entrust, Inc.", L=Ottawa,
ST=Ontario, C=CA

Signature algorithm: SHA512withRSA, 4096-bit key

[certificate is valid from 10/4/22, 5:22 PM to 1/1/29, 12:00 AM]

X.509, CN=Entrust Time Stamping CA - TS2, O="Entrust, Inc.", C=US

Signature algorithm: SHA512withRSA, 4096-bit key

[certificate is valid from 5/7/21, 7:22 PM to 12/29/40, 11:59 PM]

X.509, CN=Entrust Code Signing Root Certification Authority - CSBR1, O="Entrust,
Inc.", C=US

Signature algorithm: SHA256withRSA, 4096-bit key
```

```
[certificate is valid from 5/7/21, 3:43 PM to 11/7/30, 4:13 PM]
```

```
jar signed.
```

```
The signer certificate will expire on 2024-08-16.
```

```
The timestamp will expire on 2029-01-01.
```

## 8 Signing with REST clients

See below for using different REST clients to consume the API of the Entrust Signing Automation Service.

- [Signing data with curl commands](#)
- [Signing data with Postman](#)
- [Swagger API reference](#)

**i** See [Getting API credentials](#) for obtaining the required credentials.

### Signing data with curl commands

The following sections illustrate a step-by-step data signing with the `curl` command-line tool and the REST API of the Signing Automation Service.

- [Setting environment variables to sign with curl](#)
- [Getting an authentication token with curl](#)
- [Getting a virtual token identifier with curl](#)
- [Getting a signing key identifier with curl](#)
- [Generating a data digest](#)
- [Signing a data digest with curl](#)

**i** See [Identity Provider service](#) and [Raw Signature service](#) for a description of the exposed services.

### Setting environment variables to sign with curl

For ease of use, initialize environment variables with the API credential details previously obtained in [Getting API credentials](#).

Environment variable	API credential details
SUBJECT_ID	Subject ID
PASSWORD	Password
PARTITION	Organization ID
SIGNING_SERVICE_URL	RAW Signer URL
IDP_URL	Identity Provider (Id) URL

For example:

```
$ export SUBJECT_ID=a08*****e57b
$ export PASSWORD=1570b*****72c
$ export PARTITION=intsasapi1
$ export SIGNING_SERVICE_URL=https://rawsigner.csaas.entrust.net
```

```
$ export IDP_URL=https://idp.csaas.entrust.net
```

## Getting an authentication token with curl

Send a request to the [Identity Provider service](#) to get an [Authentication token](#).

```
$ JWT="$(curl -s -X POST -u "$SUBJECT_ID:$PASSWORD" "$IDP_URL/idp/v1/tokens" | jq -r .)" && export JWT
```

⚠ Authentication tokens have a two-minute validity, so you must periodically rerun this command.

Check the token contents – for example:

```
$ echo $JWT  
eyJhbGciOiJSUzI1NiIsImtpZCI6InBraWh1YmIkcDE1NTg5NDA3NjkiLCJ0eXAiOiJKV1QiLCJpYyQiOiJ1bnRlcnR1cm91dCI6dXN1cm91dCJ9
```

When the token generation fails, you get an error like the following.

```
{ "code": 401, "message": "Unauthorized" }
```

## Getting a virtual token identifier with curl

Authenticate with the generated [Authentication token](#) in the [Raw Signature service](#) to list the [Virtual Token identifiers](#). For example:

```
$ curl -s -H "Authorization: $JWT" "$SIGNING_SERVICE_URL/raw/partitions/$PARTITION/tokens/" | jq .  
{  
  "result": {  
    "tokens": [  
      "token1", "token2"  
    ]  
  }  
}
```

Set the desired token identifier as an environment variable – for example:

```
$ export TOKEN=token1
```

Optionally, get detailed information on the virtual token – for example:

```
$ curl -s -H "Authorization: $JWT" "$SIGNING_SERVICE_URL/raw/partitions/$PARTITION/tokens/$TOKEN" | jq .  
{  
  "result": {
```



```
"id": "token1",  
  "maxKeys": 999  
}
```

## Getting a signing key identifier with curl

Send a request to the [Raw Signature service](#) to list the identifiers of the signing keys within the Virtual Token – for example:

```
$ curl -s -H "Authorization: $JWT" "$SIGNING_SERVICE_URL/raw/partitions/$PARTITION/  
tokens/$TOKEN/keys?quantity=10" | jq .  
{  
  "result": {  
    "keys": [  
      {  
        "attributes": {},  
        "id": "cbfb2d79d86d92e3c07fabbd21c405783b4ef50f",  
        "keyParams": {  
          "type": "RSA2048"  
        }  
      },  
      {  
        "attributes": {},  
        "id": "337aeef5c25121ce7d5c9f1167387140bb60bccb",  
        "keyParams": {  
          "type": "RSA2048"  
        }  
      },  
      {  
        "attributes": {},  
        "id": "9b36fc631ac2cbff42f8b77927b838e27b12e68c",  
        "keyParams": {  
          "type": "RSA2048"  
        }  
      },  
      {  
        "attributes": {},  
        "id": "84d4cb379bc776f218142419afa23aadf56a362d",  
        "keyParams": {  
          "type": "RSA2048"  
        }  
      },  
      {  
        "attributes": {},  
        "id": "47257fe14fc41c6a8b9334d6caef80e01b2bedd6",  
        "keyParams": {  
          "type": "RSA2048"  
        }  
      }  
    ]  
  }  
}
```

```

    "attributes": {},
    "id": "600adc8742fe1e8b8816587ff97f3f43b2e30e10",
    "keyParams": {
      "type": "RSA2048"
    }
  },
  {
    "attributes": {},
    "id": "13ba033a3a9a1e8d926ae504d3de5a45c8dd4610",
    "keyParams": {
      "type": "RSA2048"
    }
  },
  {
    "attributes": {},
    "id": "5752d6b945024d344ec22ad73eb6bb5f9a018e67",
    "keyParams": {
      "type": "RSA2048"
    }
  }
],
"lastKey": "1701266768009109910"
}

```

Set the desired key identifier as an environment variable - for example:

```
$ export KEY_ID=cbeeb264a4aa3db78b8abedba6dae7fcf3548c29
```

Optionally, get detailed information on the signing key – for example:

```

$ curl -s -H "Authorization: $JWT" "$SIGNING_SERVICE_URL/raw/partitions/$PARTITION/
tokens/$TOKEN/keys/$KEY_ID" | jq .
avid@myPKIHub:~/git/src/entrust.com/chinerd/sas-clients$ curl -s -H "Authorization:
$JWT" "$SIGNING_SERVICE_URL/raw/partitions/$PARTITION/tokens/$TOKEN/keys/$KEY_ID" |
jq .
{
  "result": {
    "attributes": {},
    "value": {
      "publicKeyObject": {
        "keyType": "RSA2048",
        "value": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCA...../
QkQzGXRskrpLmJLmQIDAQAB"
      }
    }
  },
  "id": "cbfb2d79d86d92e3c07fabbd21c405783b4ef50f"
}

```

## Generating a data digest

Generate a base64-encoded hash of the data to be signed, and set this value as an environment variable. For example, to hash the contents of a `data.txt` file:

```
$ export DIGEST_TBS=$(openssl sha256 -binary ./data.txt | base64)
$ echo $DIGEST_TBS
X/D/4z204pWnLaE+7F2x0aJEWpMisoYNEQLYdqFqh7E=
```

## Signing a data digest with curl

Send a request to the [Raw Signature service](#) to sign the `$DIGEST_TBS` hash using the key with the `$KEY_ID` identifier – for example:

```
$ curl -s -H "Authorization: $JWT" "$SIGNING_SERVICE_URL/raw/partitions/$PARTITION/
tokens/$TOKEN/keys/$KEY_ID/signatures" -X POST -H "Content-type: application/json" -d
'{"hashType": "'SHA256'", "hash": "'$DIGEST_TBS'"}' | jq .
{
  "result": "aJEpAW1sR4Xu9CoFVNhYBEQcKqawonuL7Mr0e6P5Emov03YzLp1nSFmiSYgYPA/
DmBAihTESm6StVKaTevAB58WhiQoj+Eh4gPbN0SFxJPYkNP8BzMorHREAacMLqHEMwcoJM9st7M..."
}
```

## Signing data with Postman

The following sections illustrate step-by-step data signing using the Postman REST client and the REST API of the [Raw Signature service](#).

- [Importing the Raw Signature service specification in Postman](#)
- [Configuring the Postman collection](#)
- [Getting a virtual token identifier with Postman](#)
- [Getting a signing key identifier with Postman](#)
- [Signing a data digest with Postman](#)

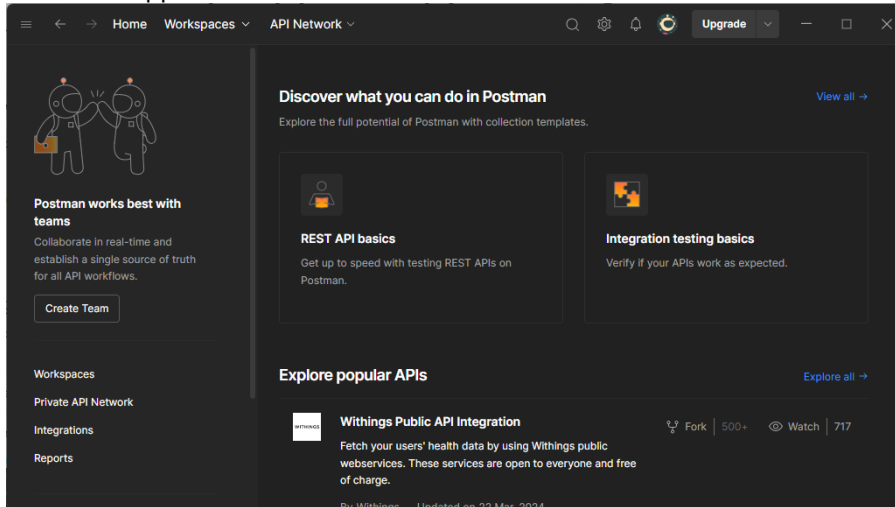
## Importing the Raw Signature service specification in Postman

Import the swagger specification of the [Raw Signature service](#) as a Postman collection.

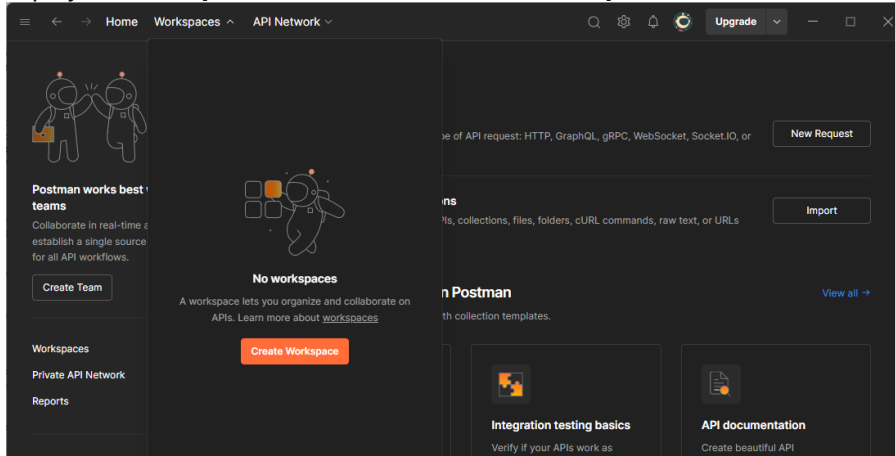
### To import the Raw Signature service specification

1. Download Postman from <https://www.postman.com/downloads>
2. Follow the installer instructions to install the application and create a user account.

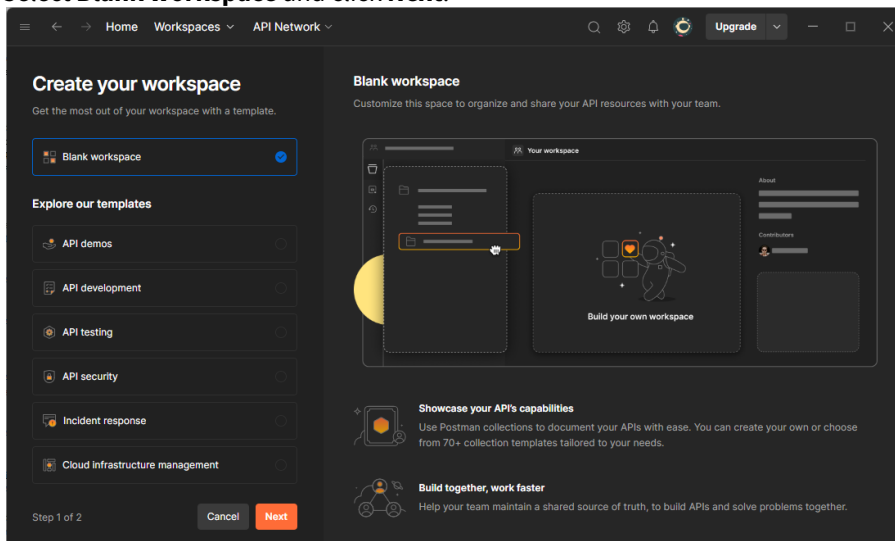
3. Launch the application.



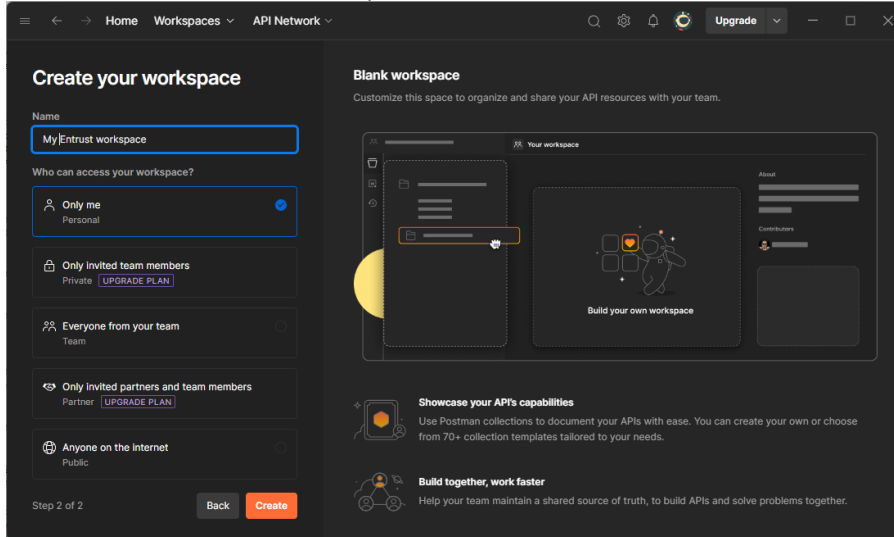
4. Deploy the **Workspaces** menu and click **Create Workspace**.



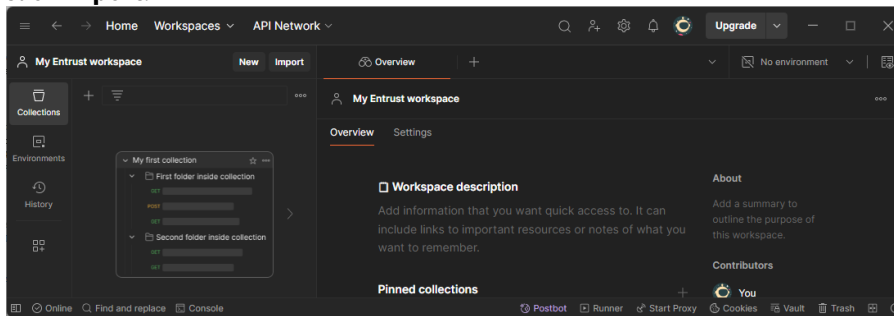
5. Select **Blank workspace** and click **Next**.



- Write the name of the new workspace and click **Create**.



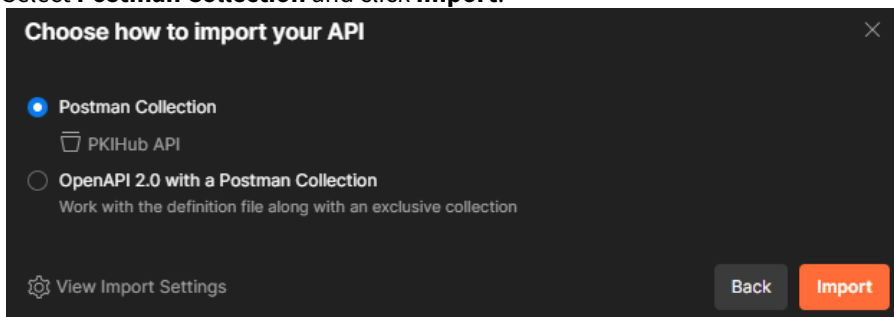
- Click **Create Workspace**.
- Click **Import**.



- Paste the following URL.

```
https://api.managed.entrust.com/sas/rawsigner-api/swagger-rawsigner-api.json
```

- Wait while Postman loads the specification.
- Select **Postman Collection** and click **Import**.

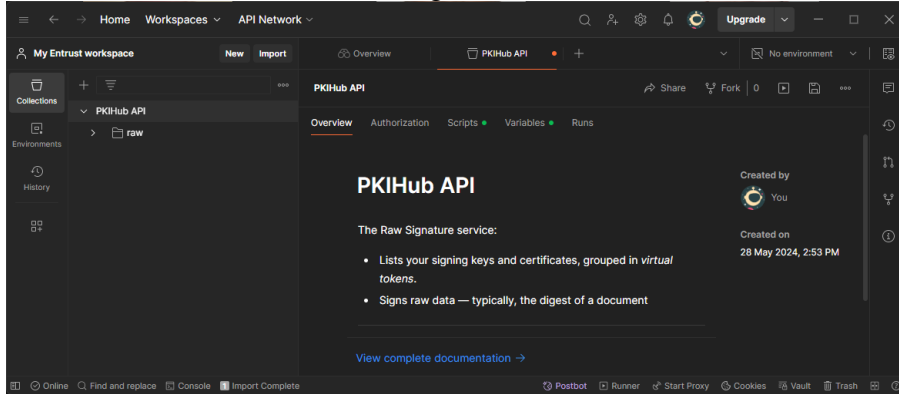


## Configuring the Postman collection

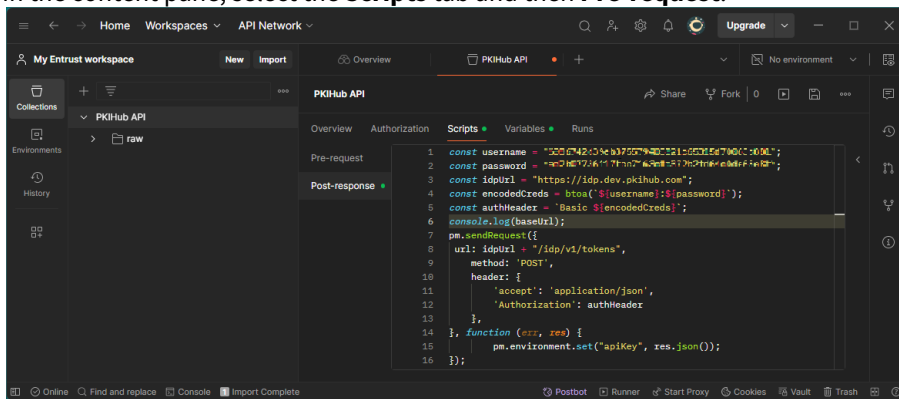
Configure the global settings of the imported **PKIHub** collection.

### To configure the PKIHub collection

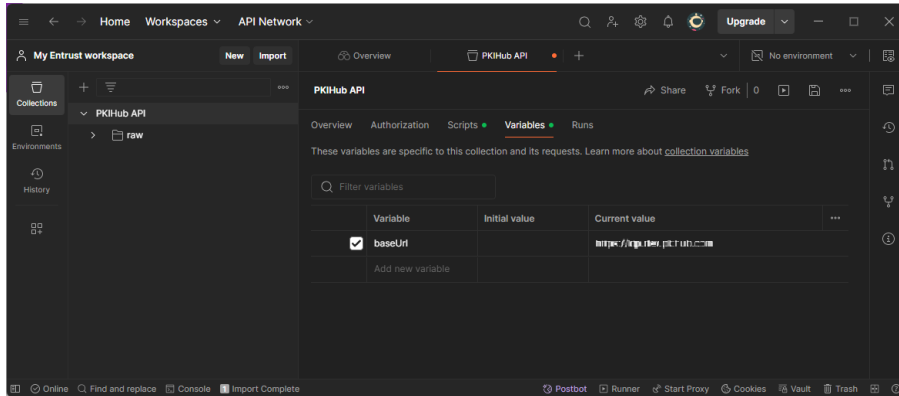
1. Click the **PKIHub** root node of the navigation tree.



2. In the content pane, select the **Scripts** tab and then **Pre-request**.



3. Paste the **Pre-request authorization script** described below.
4. Select the **Variables** tab.



5. Set the **baseUrl** variable values described below.
6. Click the **Save** icon to make the changes persistent.

### Pre-request authorization script

Add the following pre-request script to generate an **Authentication token** before running any request.

```

const username = "<subjectId>";
const password = "<password>";
const idpUrl = "<idpUrl>";

```

```
const baseUrl = pm.variables.get('baseUrl');
const encodedCreds = btoa(`${username}:${password}`);
const authHeader = `Basic ${encodedCreds}`;
console.log(baseUrl);
pm.sendRequest({
  url: idpUrl + "/idp/v1/tokens",
  method: 'POST',
  header: {
    'accept': 'application/json',
    'Authorization': authHeader
  },
}, function (err, res) {
  pm.environment.set("apiKey", res.json());
});
```

Where each placeholder corresponds to the following value.

<value>	Value
<subjectId>	The <b>Subject ID</b> value obtained in <a href="#">Getting API credentials</a> .
<password>	The <b>Password</b> value obtained in <a href="#">Getting API credentials</a> .
<idpUrl>	The <b>Identity Provider (Id) URL</b> value obtained in <a href="#">Getting API credentials</a> .

## baseUrl variable

Initialize as follows the `baseUrl` variable.

Variable	Initial value	Current value
baseUrl	Clear this value and leave the column empty	Paste the <b>RAW Signer URL</b> value obtained when <a href="#">Getting API credentials</a> .

## Getting a virtual token identifier with Postman

To get the identifier of the token holding the signing key, select the following request.

```
PKIHub API / raw / partitions / {partid} / tokens / List Virtual Tokens
```

## Params

In the content pane, select the **Params** tab to configure the following settings.

Type	Key	Value
Query Params	quantity	Leave the default value.
Query Params	lastKey	Uncheck this param to disable it.
Path Variables	partId	Paste the <b>Organization ID</b> value obtained in <a href="#">Getting API credentials</a> .

Click **Save** to make the changes permanent.

## Result

Click **Send** to send the request. The **Body** tab of the results pane will display a response like the following.

```
{
  "result": {
    "tokens": [
      "token1", "token2"
    ]
  }
}
```

## Getting a signing key identifier with Postman

To get the identifier of a signing data key, select the following request.

```
PKIHub API / raw / partitions / {partid} / tokens / {tokenid} / keys / List Keys
```

## Params

Select the **Params** tab of the content pane to configure the following settings.

Type	Key	Value
Query Params	quantity	Leave the default value.
Query Params	lastKey	Uncheck this param to disable it.
Path Variables	partId	Paste the <b>Organization ID</b> value obtained in <a href="#">Getting API credentials</a> .



Type	Key	Value
Path Variables	tokenid	One of the token identifiers previously obtained in <a href="#">Getting a virtual token identifier with Postman</a> .

Click **Save** to make the changes permanent

## Result

Click **Send** to send the request. The **Body** tab of the results pane will display a response like the following.

```
{
  "result": {
    "keys": [
      {
        "attributes": {},
        "id": "cbfb2d79d86d92e3c07fabbd21c405783b4ef50f",
        "keyParams": {
          "type": "RSA2048"
        }
      },
      {
        "attributes": {},
        "id": "337aeef5c25121ce7d5c9f1167387140bb60bccb",
        "keyParams": {
          "type": "RSA2048"
        }
      },
      {
        "attributes": {},
        "id": "9b36fc631ac2cbff42f8b77927b838e27b12e68c",
        "keyParams": {
          "type": "RSA2048"
        }
      },
      {
        "attributes": {},
        "id": "84d4cb379bc776f218142419afa23aadf56a362d",
        "keyParams": {
          "type": "RSA2048"
        }
      },
      {
        "attributes": {},
        "id": "47257fe14fc41c6a8b9334d6caef80e01b2bedd6",
        "keyParams": {
          "type": "RSA2048"
        }
      },
      {
        "attributes": {},

```

```

        "id": "600adc8742fe1e8b8816587ff97f3f43b2e30e10",
        "keyParams": {
            "type": "RSA2048"
        }
    },
    {
        "attributes": {},
        "id": "13ba033a3a9a1e8d926ae504d3de5a45c8dd4610",
        "keyParams": {
            "type": "RSA2048"
        }
    },
    {
        "attributes": {},
        "id": "5752d6b945024d344ec22ad73eb6bb5f9a018e67",
        "keyParams": {
            "type": "RSA2048"
        }
    }
],
    "lastKey": "1701266768009109910"
}

```

## Signing a data digest with Postman

To sign data with Postman, select the following request.

```

PKIHub API / raw / partitions / {partid} / tokens / {tokenid} / keys / {keyid} /
signatures / Sign Digest

```

### Params

Select the **Params** tab of the content pane to configure the following settings.

Type	Key	Value
Path Variables	partid	Paste the <b>Organization ID</b> value obtained in <a href="#">Getting API credentials</a> .
Path Variables	tokenid	One of the token identifiers previously obtained in <a href="#">Getting a virtual token identifier with Postman</a> .
Path Variables	keyid	One of the key identifiers previously obtained in <a href="#">Getting a signing key identifier with Postman</a> .

## Body

Select the **Body** tab of the content pane to configure the following parameters.

Parameter	Value
hash	The hash value of the data to be signed. See <a href="#">Generating a data digest</a> for how to generate these data.
hahsType	The identifier of the algorithm used to hash the data.

For example:

```
{
  "hashType": "SHA256",
  "hash": "8sobtsfpB9Btr+Roflefznazfk6Tt2BQItpS5szCb9I="
}
```

Click **Save** to make the changes permanent

## Result

Click **Send** to send the request. The **Body** tab of the results pane will display a response like the following.

```
{
  "result":
  "a6hk7XF9nm1TtSB6gL6BY5siytcFjUM+6xU5wSfiFHsbyWUhenQxqUFvIlW0200i0CyMLAICjWFeIksxUZkk
  VvLYz3TCV1THMoiBeVrQRpihMdjNw2DtKK2czdBWdjfttQGh1RmdYVs8eNTbGt6t1mcMEjPBw569ks6NlkC/
  aN2j3nK25N0lm6YFCZtfIasuyFuERb8JerEyx1nS7UE2RIVn3HnyIraITjKZ5mRZXvNcwqT1ptZ4tTdfwjPMD
  8vyLPWCCSxVvCy4US+fuJvkgmIjZ21u+DiQfSYGxcDXVIgPFRpiQ3b3GCPKuJC9J0y25R5ZPhCSgJ0uz9gMa9
  k8oQ=="
}
```

## Swagger API reference

See below the Swagger specification of each service exposed by the Signing Automation Service API.

Service overview	Service Swagger reference
<a href="#">Identity Provider service</a>	<a href="https://api.managed.entrust.com/sas/idp-api">https://api.managed.entrust.com/sas/idp-api</a>
<a href="#">Raw Signature service</a>	<a href="https://api.managed.entrust.com/sas/rawsigner-api">https://api.managed.entrust.com/sas/rawsigner-api</a>

## 9 Debugging the Signing Automation Client

See below for managing the debugging mode.

- [Enabling the debugging mode](#)
- [Disabling the debugging mode](#)

### Enabling the debugging mode

As explained in [Installing the Signing Automation Client](#), the out-of-the-box distribution includes the `pkcs11-logger` third-party library for debugging purposes. In debugging mode, this library logs PKCS #11 operations by sitting between the Signing Application Client and the `P11SigningClient64` library.

1. The Signing Application Client sends the PKCS #11 operation calls to the `pkcs11-logger` library.
2. The `pkcs11-logger` library redirects the calls to the `P11SigningClient64` library.
3. The `pkcs11-logger` library returns the call results to the Signing Application Client.

#### To enable the debugging mode of the Signing Automation Client

1. Rrun the `signingclient config set` command and use the `pkcs11-logger` library path as value of the `--library` option. For example:


```
signingclient config set --library "C:\Program Files\Entrust\SigningClient\pkcs11-logger64.dll"
```

2. Set the following environment variables.
  - `PKCS11_LOGGER_LIBRARY_PATH`
  - `PKCS11_LOGGER_LOG_FILE_PATH`
  - `PKCS11_LOGGER_FLAGS`

#### PKCS11\_LOGGER\_LIBRARY\_PATH


The path of the `P11SigningClient64.dll` library. As explained in [Installing the Signing Automation Client](#), the default path of this library is the following.

```
C:\Program Files\Entrust\SigningClient\P11SigningClient64.dll
```

 Do not enclose the path with quotes.

#### PKCS11\_LOGGER\_LOG\_FILE\_PATH

Set the value of this environment variable to the path of the file where logs will be recorded.

 Do not enclose the path with quotes.


## PKCS11\_LOGGER\_FLAGS

The sum of requested logging flags.

Hex	Dec	Flag
0x01	1	Disables logging into the log file
0x02	2	Disables logging process identifiers.
0x04	4	Disables logging thread identifiers.
0x08	8	Enables logging personal identification numbers (PINs).
0x10	16	Enables logging to the standard output (stdout).
0x20	32	Enables logging to the standard error destination (stderr).
0x40	64	Enables reopening of the log file. This feature decreases performance but allows deleting the log file when needed.

For example, a value of 6 disables logging:

- The process identifier.
- The thread identifier.

 When not set, this variable defaults to 0.

## Disabling the debugging mode

To disable the debugging mode, run the `signingclient config set` command again, this time using the PKCS #11 default library path as value of the `--library` option. For example:

```
signingclient config set --library "C:\Program  
Files\Entrust\SigningClient\P11SigningClient64.dll"
```

See [Installing the Signing Automation Client](#) for the folder containing the default PKCS #11 library.