TRIDENT™
TRusted IDENTity

Identity&SignatureProvider APIs

6 June, 2026

Table of Contents

- First Steps with TRIDENT APIs
- Auth 2.0 Authorization and OpenID Connect Authentication API
- Cloud Signature Consortium API
- Digital Signature API
- Document Signature API

1 First Steps with TRIDENT APIs

This document provides a practical introduction to the TRIDENT Authentication and Digital Signature APIs. Its purpose is to help developers understand the basic integration concepts, API invocation sequences, authorization flows, and signature processes through a set of guided examples and video walkthroughs.

The accompanying videos demonstrate common integration scenarios using the Authentication APIs, Cloud Signature Consortium (CSC) APIs, Digital Signature APIs, and Document Signature APIs. The examples focus on illustrating common use cases and helping developers become familiar with the overall integration model of the platform.

Along with this document, a Postman Collection is provided containing the same API calls demonstrated in the videos. Developers can use this collection to reproduce the examples, explore the APIs, and accelerate their integration efforts.

The delivery package also includes the complete Integration APIs documentation, which provides detailed information about all available endpoints, request and response formats, authentication mechanisms, authorization flows, parameters, error codes, and configuration options. While this document focuses on practical examples, the Integration APIs documentation should be considered the primary technical reference for implementation.

It is important to note that the flows presented in the videos are only examples intended to facilitate understanding of the platform capabilities. Depending on the specific business requirements and use cases, multiple integration approaches may be available. The examples were intentionally selected to demonstrate straightforward and commonly used scenarios that provide a good starting point for understanding the authentication and digital signature services available within the TRIDENT platform.

1.1 Important Notes

The APIs, examples, and videos included in this package are based on the TRIDENT Sandbox environment. The Sandbox may differ from production environments in several aspects.

In particular, the available test users, signing credentials, certificates, identity attributes, claims, and data returned by certain APIs may not exactly match what will be available in a production deployment. The specific user attributes, authentication methods, signing identities, authorization policies, and available services ultimately depend on the configuration and business requirements of each production environment.

As a result, developers should use the examples and returned data shown in the videos as illustrative references only. Integrations should be designed to support configurable user attributes, claims, credentials, and authorization models rather than relying on specific values returned by the Sandbox environment.

1.2 TRIDENT Integration Portal

This material represents only an initial introduction to the platform and its integration capabilities.

All of the content included in this package, together with additional technical documentation, API references, integration guides, tutorials, best practices, sample code, SDK documentation, onboarding material, and future resources, will be available through the TRIDENT Integration Portal.

The Integration Portal will serve as the central location for developers and integration partners to access the latest documentation, tools, API collections, technical guidance, and integration resources required to successfully integrate with TRIDENT services.

We recommend reviewing the videos, executing the provided Postman Collection, and consulting the Integration APIs documentation together to gain a complete understanding of the available integration options and integration best practices.

2 Auth 2.0 Authorization and OpenID Connect Authentication API

2.1 OAuth2 / OpenID Connect API Integration Demo Script

In this demonstration, we will show how an application can securely integrate with APIs using OAuth2 and OpenID Connect.

First, the application initiates an authorization request against the Identity Provider.

The user is redirected to the authentication page, where they enter their credentials and complete the login process. Once authentication is successful, the Identity Provider returns an authorization code to the application.

Next, the application exchanges this authorization code for an OAuth2 access token by calling the token endpoint.

As a result, the application receives a Bearer access token. This token represents the authorization granted to the application and contains the scopes that define what information and resources can be accessed.

2.2 Example 1: Basic Profile Access

In the first example, the application requests a token with the `profile` scope.

Using this token, the application calls the User Information API.

The API validates the access token and checks the authorized scopes before returning the response.

Because the token only contains the `profile` scope, the API returns a limited set of user attributes, such as:

```
{
  "name": "Federico Seijo",
  "given_name": "Federico",
  "family_name": "Seijo"
}
```

This demonstrates the principle of least privilege: the application only receives the information that has been explicitly authorized through the requested scope.

2.3 Example 2: Extended Identity Information

Next, we repeat the process, but this time requesting additional scopes, such as:

```
openid full_identity
```

A new access token is issued with broader permissions.

When the application calls the same User Information API using this token, the response contains a richer set of identity attributes:


```
{
  "serialNumber": "BB_NID_61112222",
  "given_name": "Federico",
  "family_name": "Seijo",
  "phone_number": "0961112222",
  "email": "federico.seijo@interfase.global",
  "registration_level": "2"
}
```


The API dynamically determines which attributes can be returned based on the scopes present in the access token.

2.4 Summary

This demonstration highlights how OAuth2 and OpenID Connect provide secure and controlled access to user information.

- OAuth2 protects API access through access tokens.
- OpenID Connect extends OAuth2 with identity capabilities.
- Scopes define which user attributes and resources an application can access.
- APIs validate the token and enforce authorization policies before returning any data.
- Different scopes result in different levels of information disclosure, allowing organizations to implement privacy-by-design and least-privilege principles.

This approach enables secure integration between applications and identity services while ensuring that user information is shared only when properly authorized.

3 Cloud Signature Consortium API

This demonstration showcases the end-to-end integration of a signature application with the TRIDENT Cloud Signature Consortium (CSC) API.

The objective is to illustrate how a third-party application can discover user signing credentials, obtain authorization from the user, and prepare a signing operation using the standardized CSC interfaces.

3.1 Step 1 – Create a Pushed Authorization Request (PAR)

The process begins by creating a Pushed Authorization Request.

The application sends a request to the CSC OAuth endpoint:

```
POST /csc/v2/oauth2/pushed_authorize
```

This request contains the authorization parameters required for the signing operation, such as:

- Client identifier
- Redirect URI
- Credential scope
- Credential identifier
- Number of signatures
- Document hashes
- Hash algorithm
- PKCE parameters

TRIDENT validates the request and returns a `request_uri`.

```
{
  "request_uri": "urn:ietf:params:oauth:request_uri:...",
  "expires_in": 60
}
```

This approach follows RFC 9126 and prevents sensitive information, such as document hashes, from being exposed through browser URLs.

3.2 Step 2 – Start User Authorization

The application then redirects the user's browser to the CSC authorization endpoint.

```
GET /csc/v2/oauth2/authorize
```

Instead of sending all authorization parameters again, the application only provides:

- client_id
- request_uri

TRIDENT retrieves the previously pushed authorization request and starts the user authorization process.

The user authenticates and reviews the requested signing operation.

3.3 Step 3 – Receive the Authorization Code

After successful authentication and consent, TRIDENT redirects the browser back to the configured redirect URI.

The authorization code is returned as part of the redirect URL.

```
https://localhost?code=...
```

The video shows this redirection taking place and the authorization code being delivered to the application.

This code represents the user's approval of the requested signing operation.

3.4 Step 4 – Exchange the Authorization Code for a Token

The application exchanges the authorization code at the CSC token endpoint.

```
POST /csc/v2/oauth2/token
```

The request includes:

- grant_type=authorization_code
- authorization code
- client identifier
- PKCE verifier (if used)

TRIDENT validates the request and issues the corresponding access token.

3.5 Step 5 – Authenticate the User and Retrieve User Information

The demonstration also includes retrieving information about the authenticated user.

Using the obtained access token, the application invokes the user information endpoint and receives the identity attributes associated with the authenticated user.

This step confirms that the authorization process was completed successfully and that the application can identify the signer.

3.6 Step 6 – Obtain a Service Access Token Using Client Credentials

The video then demonstrates a second OAuth flow based on Client Credentials.

The application requests a service token directly from TRIDENT:

```
POST /csc/v2/oauth2/token
grant_type=client_credentials
```

The request is authenticated using the application's client credentials.

TRIDENT responds with a Service Access Token.

```
{
  "access_token": "...",
  "token_type": "Bearer",
  "expires_in": 120
}
```

This token is used for backend-to-backend operations that do not require direct user interaction.

3.7 Step 7 – Retrieve the User Credential List

Using the Service Access Token, the application invokes:

```
POST /csc/v2/credentials/list
```

The request includes:

```
{
  "userID": "<user>",
  "credentialInfo": true,
  "certificates": "chain",
  "certInfo": true
}
```

TRIDENT returns the signing credentials associated with the user.

The response contains:

- Credential identifiers
- Certificate chains
- Certificate metadata
- Credential capabilities
- Supported algorithms
- Key information

This allows the application to discover which signing credentials are available for future signing operations.

3.8 Step 8 – Create the signing authorization request

After the application has already retrieved the user's credentials, it starts the signing authorization process.

The application sends a request to the CSC Pushed Authorization Request endpoint.

This request includes the information required to authorize the signature operation, such as the selected credential, the number of signatures, the redirect URI, and the PKCE parameters.

TRIDENT validates the request and returns a `request_uri`.

3.9 Step 9 – Start the user authorization flow

The application uses the `request_uri` to start the CSC OAuth2 authorization flow.

The browser is redirected to TRIDENT with the `client_id` and the `request_uri`.

TRIDENT loads the previously pushed authorization request and starts the user interaction.

3.10 Step 10 – User authenticates and authorizes the signature

The user authenticates in TRIDENT.

Then, the user authorizes the use of the selected signing credential for the requested signing operation.

This authorization is specific to the credential and the signing operation requested by the application.

3.11 Step 11 – TRIDENT returns the authorization code

After successful authorization, TRIDENT redirects the browser back to the application.

The redirect contains an OAuth2 authorization code.

The application will use this code to request the signing token.

3.12 Step 12 – Exchange the authorization code for a signing token

The application sends the authorization code to the CSC token endpoint.

The token represents the user's authorization to use the selected credential for the signing operation.

3.13 Step 13 – Generate the signature

The application invokes the CSC signature operation using the selected credential and the token.

The request includes the hash to be signed, the credential identifier, and the signature algorithm.

TRIDENT validates the token and generates the digital signature using the protected signing key.

3.14 Step 14 – Receive the signature result

TRIDENT returns the generated signature value.

The application can then use this signature to complete the final document signature or continue with its own signature container processing.

3.15 Summary

The Cloud Signature Consortium (CSC) API provides a standardized interface for integrating remote digital signature services into third-party applications. It enables applications to securely access user signing credentials, obtain user authorization for signing operations, and generate digital signatures using signing keys protected within the TRIDENT infrastructure.

The CSC API follows the Cloud Signature Consortium specifications and leverages OAuth 2.0 authorization flows to authenticate users, authorize the use of signing credentials, and protect access to signing services. This approach enables interoperability with external applications while supporting secure remote signing use cases.

4 Digital Signature API

This demonstration showcases the TRIDENT Digital Signature API and illustrates two common remote signing scenarios:

- Interactive server-side document signing
- Batch document signing

The Digital Signature API provides a simplified abstraction layer on top of the underlying signing infrastructure, allowing applications to request digital signatures without having to directly manage signing credentials, certificate operations, or cryptographic key access.

4.1 Step 1 – User Authentication

The user starts by authenticating through the OAuth 2.0 flow.

After successful authentication, the application receives an authorization code.

4.2 Step 2 – Token Exchange

The application exchanges the authorization code for an access token.

This token represents the authenticated user session and allows the application to access user-related signing resources.

4.3 Step 3 – Retrieve User Signing Identities

Using the access token, the application calls the Signing Identity API to obtain the signing identities associated with the authenticated user.

The response includes the available signing credentials that the user can use to generate digital signatures.

4.4 Step 4 – Request Signing Authorization

Before generating the signature, the application requests a new authorization specifically for the signing operation.

At this point, the user is asked to enter their signing PIN.

4.5 Step 5 – User Confirms with PIN

The user enters the PIN to authorize the use of the selected signing credential.

This step confirms the user's consent and proves control over the signing credential.

4.6 Step 6 – Exchange Authorization Code for Signing Token

After the user authorizes the operation, TRIDENT returns a new authorization code.

The application exchanges this code for a signing token.

This token authorizes the application to perform the requested signature operation.

4.7 Step 7 – Create a Digital Signature

Once authorization has been completed, the application invokes the Digital Signature API.

```
POST createDigitalSignatureOnServer
```

The request contains:

- Signing authorization token
- Document hash
- Signature format
- Additional signing options

TRIDENT validates the authorization and uses the user's remote signing credential to generate the digital signature.

The private signing key never leaves the secure signing environment.

The response contains the generated signature that can be embedded into the target document.

4.8 Step 8 – Batch Signing Workflow

The demonstration also shows the batch signing capability.

```
POST createDigitalSignatureOnServerAsBatch
```

This operation allows an application to submit multiple signing requests that will be processed as a single authorized transaction.

Instead of requesting user authorization for every individual document, the user can authorize a predefined number of signatures in advance.

The batch request may contain:

- Multiple document hashes
- Signing metadata

TRIDENT processes the signing requests using the previously granted authorization.

This significantly improves efficiency when large numbers of documents must be signed.

4.9 Summary

This demonstration shows the complete digital signing workflow using the Digital Signature API.

The process begins with user authentication through OAuth 2.0, followed by the exchange of the authorization code for an access token. Using this token, the application retrieves the signing identities available to the authenticated user and selects the credential that will be used for signing.

The application then requests a dedicated signing authorization. The user is prompted to enter their signing PIN, providing explicit consent and authorization to use the selected signing credential.

Once the authorization is approved, the application exchanges the resulting authorization code for a signing token. This token is subsequently used to invoke the Digital Signature API and generate a digital signature.

Finally, the demonstration shows both a single-signature operation and a batch-signature operation, where multiple signatures are generated using the same signing authorization, within the limits granted by the user.

5 Document Signature API

5.1 Document Signature API Demonstration

This demonstration showcases the Document Signature API, a simplified remote signing integration that allows applications to submit complete documents to TRIDENT for signing.

Unlike the Digital Signature API, where the application is responsible for generating document hashes and managing signature containers, the Document Signature API handles the entire document signing process on the server side.

5.1.1 Step 1 (*optional only for demo*) – User Authentication

The process begins with user authentication through OAuth 2.0.

The user authenticates and grants access to the application.

The application then exchanges the authorization code for an access token.

5.1.2 Step 2 (*optional only for demo*) – Retrieve User Signing Identities

Using the access token, the application retrieves the signing identities associated with the authenticated user.

The available signing credentials are displayed, allowing the application to select the credential that will be used for the signing operation.

5.1.3 Step 3 – Obtain the Signing Token

After successful authorization, the application exchanges the authorization code for a signing token.

This token authorizes the subsequent signing operation.

5.1.4 Step 4 – Upload the Document

The application uploads the complete document to the Document Signature API.

In the demonstration, a PDF document is uploaded directly to the TRIDENT signing service.

Unlike hash-based signing APIs, the document itself is transferred to the signing platform.

5.1.5 Step 5 – Server-Side Document Signing

TRIDENT receives the document and performs all signing operations internally, including:

- Document processing
- Hash calculation
- Signature generation
- Signature embedding

- Signed document generation

The application does not need to calculate hashes or construct signature containers.

5.1.6 Step 6 – Download the Signed Document

After the signing operation completes successfully, TRIDENT returns the signed PDF document.

The demonstration shows the signed document being downloaded and opened for verification.

5.1.7 Integration Considerations

The Document Signature API provides one of the simplest integration models available because all signing-related operations are handled by TRIDENT.

Benefits include:

- Faster integration.
- Reduced implementation complexity.
- No document hash calculation required.
- No PDF signature container management required.
- No cryptographic processing required on the application side.

However, organizations should consider that the complete document is transmitted to and processed by the TRIDENT signing infrastructure.

As a result, before adopting this integration model, it is important to evaluate:

- Data privacy requirements.
- Information classification policies.
- Regulatory and compliance obligations.
- Data residency requirements.
- Internal security policies regarding document transmission and storage.

For use cases where document contents cannot leave the application environment, the CSC API or Digital Signature API should be a more appropriate option because only document hashes are sent to the signing service.

5.1.8 Summary

The Document Signature API offers the simplest and fastest integration approach, while requiring organizations to consider that the document content is processed within the TRIDENT signing environment as part of the signing operation.