Atostek ID 4.4 Integration Guide

Atostek

Table of contents

1. Atostek ID Smart Card reader software

<table>
<tr><th colspan="2">Atostek ID support</th></tr>
<tr><td>Operating systems</td><td>
<table>
<tr><th></th><th>x86</th><th>x64</th><th>ARM</th></tr>
<tr><td>Windows 10</td><td>x</td><td>x</td><td></td></tr>
<tr><td>Windows 11</td><td></td><td>x</td><td></td></tr>
<tr><td>Windows Server 2016, 2019 & 2022</td><td></td><td>x</td><td></td></tr>
<tr><td>Linux Debian & RedHat</td><td></td><td>x</td><td></td></tr>
<tr><td>MacOS 12, 13, 14, 15 & 26</td><td></td><td>x</td><td>x</td></tr>
</table>
</td></tr>
<tr><td>Reader interface</td><td>PC/SC</td></tr>
<tr><td>Smart Cards</td><td>Oberthur IAS-ECC
Idemia Ideal Citiz 2.17i ID.me
Idemia Cosmo X
Gemalto MultiApp v3.0
Gemalto MultiApp v4.2
Thales MultiApp v5.0
Thales MultiApp 5.1</td></tr>
<tr><td>Cryptographic Interfaces</td><td>erasmartcard.ehoito.fi
SCS v1.1, v1.2, v1.3
PKCS#11 v3.1
Windows CryptoAPI (Minidriver)
macOS CryptoTokenKit (Tokendriver)</td></tr>
<tr><td>Cryptographic algorithms</td><td>RSA-PKCS#1 v1.5, RSA-PKCS#1 v2.0 (OAEP), RSA-PKCS#1 v2.1 (PSS)</td></tr>
<tr><td>Elliptic curve types</td><td>ECDH, ECDSA</td></tr>
<tr><td>TLS protocols</td><td>TLS 1.2 and 1.3</td></tr>
</table>

2. Modules

This section provides an overview of each module included in the Atostek ID installation package.

2.1. PKCS#11 module

Atostek ID includes its own implementation of the Public Key Cryptography Standard (PKCS) #11 interface. It adheres to version 3.1 of the specification. A detailed description of the specification is available on the OASIS portal: PKCS #11 Specification v3.1 (https://docs.oasis-open.org/pkcs11/pkcs11-spec/v3.1/pkcs11-spec-v3.1.pdf). The module is included in the Atostek ID installation package, and the locations of the module on different operating systems are listed in the table below.

PKCS#11 module location in different platforms	
Platform	Path
Windows	C:\Program Files (x86)\Atostek ID\Atostek-ID-PKCS11_Winx64.dll C:\Program Files (x86)\Atostek ID\Atostek-ID-PKCS11_Winx86.dll
MacOS	/Library/Atostek ID/Atostek-ID-PKCS11.dylib
Debian	/usr/lib/Atostek-ID-PKCS11.so
Red Hat	/usr/lib64/Atostek-ID-PKCS11.so

Detailed installation instructions are available in the installation guides. For a comprehensive description of the supported PKCS interface functions in the latest Atostek ID release, refer to the Release Notes documentation.

2.2. Cryptographic interfaces of operating systems

2.2.1. Windows Minidriver

The Atostek ID Windows installation package includes the Atostek ID Windows Smart Card Minidriver. The Minidriver implements version 7.07 of the Microsoft specification, which is described in detail on the Microsoft Developer Portal. The Minidriver is installed automatically during the setup process and requires no additional steps to activate it within the operating system. It is a Windows Hardware Quality Labs (WHQL) signed driver. Both 64-bit and 32-bit versions are installed automatically.

For Certificate-Based Authentication (CBA) changes on Windows domain controllers (Microsoft KB5014754), Atostek ID provides a separate Active Directory Registration Service (ADRS). The ADRS works together with the Minidriver to configure the smart card's certificate to the correct Active Directory (AD) user. Detailed instructions for installing and using ADRS can be found in the ADRS Installation Document.

2.2.2. MacOS Tokendriver

For macOS, the TokenDriver serves as the smart card driver. The Atostek ID Tokendriver enables the use of DVV-issued smart cards on macOS systems through the CryptoTokenKit interface. The Tokendriver supports functionalities such as smart card-based login to the system. To verify that an inserted smart card is functioning correctly with the Tokendriver, the following command can be used:

```
1. security list-smartcards
```

2.2.3. Linux p11-kit and PKCS module

The Atostek ID PKCS module can be used on Linux systems with the p11-kit to perform cryptographic operations. Detailed installation instructions for different use cases are available in the Atostek ID Linux installation guides.

2.2.3.1. GNOME-desktop smartcard login

The GNOME workbook implementation using the Atostek ID PKCS#11 module requires manual SSSD configuration. Additionally, the certificates stored on the card are required for this setup.

The process of setting the DVV root certificate and issuing certificate as trusted is demonstrated below. If needed, the certificates can be installed in a different file path. However, in such cases, the `pam_cert_db_path` parameter must be added to the SSSD configuration under the `[pam]` section.

```
DVV_ROOT_CERT_PATH=<ROOT CA PEM FILE PATH>

DVV_INTERMEDIATE_CERT_PATH=<INTERMEDIATE CA PEM FILE PATH>

1. sudo --install --directory --mode=600 /etc/sssd/pki
2. cat "$DVV_ROOT_CERT_PATH" "$DVV_INTERMEDIATE_CERT_PATH" | sudo tee
   --append /etc/sssd/pki/sssd_auth_ca_db.pem > /dev/null
```

Logging in with a local user (users found in the /etc/passwd file) is enabled as follows:

```
TARGET_USER=<TARGET USER>

COMMON_NAME=<USER AUTHENTICATION CERTIFICATE COMMON NAME>


1. cat << EOF | sudo tee /etc/sssd/sssd.conf.d/atostek-id.conf >
   /dev/null

[sssd]

enable_files_domain = True

services = pam

[certmap/implicit_files/$TARGET_USER]

matchrule = <SUBJECT>.*CN=$COMMON_NAME.*

[pam]

pam_cert_auth = True
```

More detailed instructions

https://access.redhat.com/articles/4253861

https://ubuntu.com/tutorials/how-to-use-smart-card-authentication-in-ubuntu-desktop

https://sssd.io/design-pages/matching_and_mapping_certificates.html

2.3. SCS web signature API

The SCS API is the Digital Signature Creation Service (SCS) specified by DVV. The Atostek ID package implements interface versions 1.1, 1.2 and 1.3 of the specification. Detailed documentation on the specification is available on the DVV website: https://dvv.fi/en/fineid-specifications. To facilitate easier integration with the signature service, Atostek ID provides a .NET Standard toolkit library for the SCS API.

2.3.1. Toolkit

1) Errors and Error Codes

Atostek ID Toolkit uses the class ToolkitError to represent the success or failure of an operation. It has the following members:

public int errorCode: *A numeric code representing the type of the error.*

public string errorText: *A string describing the error in more detail.*

The possible values of errorCode are the following:

Value	Meaning
0	No error
3000	Serialization failed
3100	Deserialization failed
3200	Response lacking mandatory parameters

2) Requests

The classes representing requests are the following:

SCSSignRequest

The members of the class SCSSignRequest are the following:

Type	Name	Description
string?	**version**	The version of the SCS specification that is expected to be supported
SCSSelector?	**selector**	The certificate selector parameters
string	**content**	The data to be signed, base64 encoded
string?	**contentType**	The type of the data. May be “data” or “digest”.
string?	**hashAlgorithm**	The requested signature hash algorithm.
string?	**signatureAlgorithm**	The signature algorithm.
string?	**signatureType**	The requested signature format. May be “signature”, “cms” or “cms-pades”.

The class SCSSignRequest has a constructor:

public SCSSignRequest(string? version, SCSSelector? selector, string content, string? contentType, string? hashAlgorithm, string? signatureAlgorithm, string? signatureType)

SCSSelector

The class SCSSelector does not represent a request, but it is used in the class SCSSignRequest. Its members are the following:

Type	Name	Description
List<string>?	**issuers**	A list of acceptable issuers of the end user certificate in string format or in ASN1/DER encoded form.
List<string>?	**akis**	A list of authority key identifiers of acceptable issuers in base64 format.
List<string>?	**keyusages**	A list of required keyusages.
List<string>?	**keyalgorithms**	A list of acceptable key algorithms of the end user certificate.
bool?	**strictKeypolicy**	A selector used together with keyalgorithms. If the value is true, selected end user certificate must have one of the listed key algorithms as type of the subject public key in the end user certificate.

The class SCSSelector has a constructor:

public SCSSelector(List<string>? issuers = null, List<string>? akis = null, List<string>? keyusages = null, List<string>? keyalgorithms = null, bool? strictKeypolicy = null)

3) Responses

The classes representing responses are the following:

SCSSignResponse

The members of the class SCSSignResponse are the following:

Type	Name	Description
string	**version**	The version of the SCS specification that the SCS supports
string	**status**	Whether the signature creation operation was successful. Possible values are “ok” and “failed”.
int	**reasonCode**	The reason code of the response status.
string	**reasonText**	Textual description of the reason code.
string?	**signature**	The digital signature, base64 encoded.
string?	**signatureType**	The type of the signature. May be "signature", “cms” or “cms-pades”.
string?	**signatureAlgorithm**	The format of the digital signature.
List<string>?	**chain**	The certificate chain of the end user certificate.

The class SCSSignResponse has a constructor:

public SCSSignResponse(string version, string status, int reasonCode, string reasonText)

SCSVersionResponse

The members of the class SCSVersionResponse are the following:

Type	Name	Description
string	**version**	The version of the SCS specification that the SCS supports
string	**httpMethods**	The allowed HTTP methods for the signature creation request.
string	**contentTypes**	The data types supported by the SCS.
string	**signatureTypes**	The signature types supported by the SCS.
bool	**selectorAvailable**	Whether the SCS supports the selector functionality.
string	**hashAlgorithms**	The hash algorithms supported by the SCS.
string	**signatureAlgorithm**	The signature algorithms supported by the SCS.
string?	**applicationOIDs**	The display list of object identifiers.

The class SCSVersionResponse has a constructor:

public SCSSignResponse(string version, string status, int reasonCode, string reasonText)

4) Functions

The class AtostekIDToolkit has functions with which objects of the request classes can be converted into JSON strings and with which JSON strings can be converted into objects of the response classes.

deserializeSCSVersionResponse

The function deserializeSCSVersionResponse creates an SCSVersionResponse object according to the given JSON. Its signature is the following:

public ToolkitError deserializeSCSVersionResponse(string jsonRes, out SCSVersionResponse? res)

It takes as a regular parameter the string *jsonRes*, which is the JSON that describes the response, and as an out parameter the variable *res* which is of the type SCSVersionResponse? and through which the created SCSVersionResponse object is returned. The function returns a ToolkitError object, which describes the success or failure of the operation.

The returned ToolkitError's error code can be 0 (No error) if the creation of the SCSVersionResponse object was successful, or 3100 (Deserialization failed) or 3200 (Response lacking mandatory parameters) if the creation was unsuccessful.

serializeSCSSignRequest

The function serializeSCSSignRequest creates a JSON or URL parameter string based on an SCSSignRequest object. Its signature is the following:

public ToolkitError serializeSCSSignRequest(SCSSignRequest req, out string jsonReq, HttpMethod method = HttpMethod.HTTPPOST)

It takes as a regular parameter an SCSSignRequest object, *req*, according to which the JSON or URL parameter string is formed; as an out parameter the string variable *jsonReq* through which the formed string is returned; and as another regular parameter the HttpMethod enum *method*, which determines whether the result shall be a JSON string or a URL parameter string. HTTPPOST leads to a JSON string while HTTPGET leads to a URL parameter string. The function returns a ToolkitError object, which describes the success or failure of the operation.

The returned ToolkitError's error code can be 0 (No error) if the creation of the result string was successful, or 3000 (Serialization failed) if the creation was unsuccessful.

deserializeSCSSignResponse

The function deserializeSCSSignResponse creates an SCSSignResponse object according to the given JSON. Its signature is the following:

public ToolkitError deserializeSCSSignResponse(string jsonRes, out SCSSignResponse? res)

It takes as a regular parameter the string *jsonRes*, which is the JSON that describes the response, and as an out parameter the variable *res* which is of the type SCSSignResponse? and through which the created SCSSignResponse object is returned. The function returns a ToolkitError object, which describes the success or failure of the operation.

The returned ToolkitError's error code can be 0 (No error) if the creation of the SCSSignResponse object was successful, or 3100 (Deserialization failed) or 3200 (Response lacking mandatory parameters) if the creation was unsuccessful.

2.4. erasmartcard.ehoito.fi HTTPS signature API

The erasmartcard.ehoito.fi HTTPS signature API is a custom authentication and signature interface developed for Atostek ID. This interface extends the functionality and support provided by the SCS HTTPS Signature API. For more details about this interface, please contact Atostek.

2.4.1. erasmartcardpost:// -protocol usage from a browser

Atostek ID responds to the following protocols:

- erasmartcardpost:
- erasmartcardpost://

Following the protocol, the address to which the POST request is sent is specified. The parameters in the address define all the details that will be included in the POST request from Atostek ID. The `{PORT}` placeholder in the parameters is replaced with the port information of the erasmartcard.ehoito.fi service used by Atostek ID. The POST protocol in Atostek ID supports both XML and JSON message formats. The format is determined by specifying the `type` parameter in the request. JSON is the default format.

The link opened in the browser therefore consists of the following parts:

- Post protocol: `erasmartcardpost:` or `erasmartcardpost://`
- The URL where the POST request is sent (for example): `https://www.esimerkki.fi/PortData`
- The parameters that are sent in the POST request as such and the following must be taken into account:
    - Parameters are separated from URL with a `?` sign

- Parameters are separated by `&`
- Parameters and values are separated using the `=` sign
- The `type` parameter tells the type of the POST request
  - possible values: `json` or `xml`
  - `json` by default
- The `{PORT}` embedding is replaced with the port information of the Atostek ID

Pattern: `<post-protocol><url>?<parameter1>=<parameter1 value>&<parameter2>=<parameter2 value>`

The following should be noted about the POST request sent by Atostek ID:

- Headers are set to `Content-Type` either `application/json` or `application/xml` depending on the value of `the type` parameter of the request
- The XML request message is placed inside the `<result></result>` element.
- If `https://` or `http://` is not provided, URLs default to https `://` .
- If the request fails, the transmission will be retried 10 times from the Atostek ID. If the request is still not successful after this, Atostek ID stops trying to send the request. Waits 2 seconds after a failed request.

Example 1:

A link to be opened from a browser:

```
erasmartcardpost://https://www.esimerkki.fi/
PortData?type=json&eraScPort={PORT}&userId=1234321
```

Request from Atostek ID:

```
POST https://www.esimerkki.fi/PortData

Header: Content-Type: application/ json

{
        "eraScPort": "44304",
        "userId": "1234321"
}
```

Example 2:

A link to be opened from a browser:

```
erasmartcardpost://https://www.esimerkki.fi/
PortData?type=xml&eraScPort={PORT}&userId=1234321
```

Request from Atostek ID:

```
POST https://www.esimerkki.fi/PortData

Header: Content-Type: application/xml

<result>

       <eraScPort>44304</eraScPort>
       <userId> 1234321</userId>

</result>
```