Parakstšanas API (SignAPI)

Parakstšanas API (SignAPI)

Paketes kopsavilkuma aprinšanas piemrs

Scope

Provide a signable data value calculation from the signable file by using the end-user signature certificate received.

Description

1. Start signing session. If there are problems with the files or the request does not meet the format requirements for the document signing, the process is terminated and an error is returned;
2. The hash of each file to be signed are calculated as well as signable data;
3. The signature algorithm is determined from certificate;
4. The prepared signable data in *base64* format is returned to the requester for signature.

Request

The Service provider's application sends the following GET request using TLS:

```
POST /api-sign/v1.0/CalculateDigest
```

Authorization

The request must contain an *Authorization* header with an OAuth **Introspect access token** obtained via Integration Platform a Service provider's credentials grant flow.

Introspect piekuves talona (token) saemšana

Description

This operation obtains an *OAuth 2.0* access token. This operation can be invoked as part of an OAuth 2.0 Service provider's credentials grant flow.

Introspect access token

When the Service provider's credentials grant flow is used, the obtained access token demonstrates the administrative authorization of the S ervice provider's application making the call for accessing certain resources or services (i.e., without direct intervention of the resource's owner), or for accessing resources of the Service provider's application. Token is issued when the authorization server that processes the request is not associated to an identity provider. A token of this type can be used for accessing resources not associated to end-users or to end-user resources of any domain.

This type of access token is used to get access to Signature creation and validation service API's

Request

To obtain the token, the Service provider's application must send a request like the following to authorization server using TLS. This request is sent directly from the Service provider’s application to authorization server and does not go via the browser.

```
POST /trustedx-authserver/oauth/{as}/token
```

Parameter

Title	Type	Field	Description
as	path	mandatory	Use "lvrtc-eipsign-as"

Host:

Test environment: eidas-demo.eparaksts.lv

Production: eidas.eparaksts.lv

Content-Type Header

```
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
```

In HTTP POST request is necessary to incorporated the following main attribute: *Authorization* – API access token.

```
Authorization: Basic <API-Key>
```

How to generate API Access Key

Before Service provider access Integration platform API, LVRTC shall register Service provider as customer of Integration platform. After signing a contract with LVRTC (Test of Production environment) LVRTC generates Service Provider's application identifier – (*client_id*) and shared secret (*client_secret*), intended for the customer usage.
API Access Key (API Key) is generated from the Service provider's application identifier (*client_id*), a secret shared with the platform (*client_secret*) on the following basis:

Service provider's application identifier *client_id* are converted using the UTF-8 character encoding and URL encoding conditions.

For example, value "Portls" conversion result is "port%C4%81ls".

Service provider's application password *client_secret* is converted by using the UTF-8 character encoding and URL encoding conditions.

For example, value "drošba" conversion result is "dro%C5%A1%C4%ABba".

Both values of the previous two steps must be combined with separator colon ":" between them.

For example, by using previous examples, the result will be "port%C4%81ls:dro%C5%A1%C4%ABba".

Obtained value must be converted using base 64 encoding without line breaks.

For example, values "port%C4%81ls:dro%C5%A1%C4%ABba" conversion result is "CG94ydCVDNCUMWxzOmRybyVDNSVBMSVDNCVBQmJh".

"MIME Tools" tool in "Notepad + +" can be used for this purpose.

```
API-Key = base64[url_encode(utf8(<client_id>)) ':' url_encode(utf8(<client_secret>))]
```

Body

The content of the request for Introspect access token (used for access SignAPI service):

Property	Usage	Description
`grant_type`	mandatory	Must have the *client_credentials* value .
scope	mandatory	Must have the *urn:safelayer:eidas:oauth:token:introspect* value

Example (Introspect access token)

The following example shows a situation in which the Service provider's application with the identifier "*Portal*" and the password "*drošba*" authority shall transmit the request to the server with the identifier "*lvrtc-eips-as*":

```
POST /trustedx-authserver/oauth/lvrtc-eipsign-as/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic CG94ydCVDNCUMWxzOmRybyVDNSVBMSVDNCVBQmJh
Host: eidas-demo.eparaksts.lv
grant_type=client_credentials&
                        scope=urn%3Asafelayer%3Aeidas%3Aoauth%3Atoken%3Aintrospect
```

Response

In response, Integration platform authorization server issues a bearer-type OAuth 2.0 access token and returns it in a JSON structure.

```
{
"access_token" : {string},
"token_type" : "Bearer",
"expires_in" : {number}
}
```

Parameter

Property	Description
`access_token`	Access token generated by Authorization server. The token has the characteristics specified in the configuration of the authorization server that processed the request and consists of a random string of the number of bytes specified in the **Access token number of random bytes** (by default, 32), encoded in hexadecimal.
`token_type`	Type of access token. Always has the "*bearer*" value. (Bearer type OAuth 2.0 access token).
`expires_in`	Lifetime (in seconds) of the access token. The Service provider's application must perform the access the token authorizes before the token expires. This value can be configured in the **Token timeout** option of the authorization server (by default, 120 seconds). Once this timeout has expired, the token becomes invalid, and the Service provider's application must obtain another one if it wants to continue invoking the protected services.

Example

Introspect access token:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
{
 "scope": "urn:safelayer:eidas:oauth:token:introspect",
 "access_token": "dfffb0d7f90bed142464750cacad5e4b9e23f58ecb1d77e3bdf706ba208ad16a",
 "token_type": "Bearer",
 "expires_in": 600
}
```

Body

```
{
"sessions":[
    {
      "sessionId": "string"
    }
  ],
  "certificate": "string",
  "signAsPdf": true,
  "createNewEdoc": true
}
```

Property	Description
`sessions`	Information about file processing sessions
`sessions.sessionId`	File processing session identifier. It is possible to specify multiple sessions.
`certificate`	Signing certificate in *base64* format
`signAsPdf`	**`True`** - will be signed as PDF (can only be used for signing PDFs and only one PDF file per session). In this case, the propery "createNewEdoc" will be ignored. **`False`** - will create XAdES signature in ASICE container (EDOC).
`createNewEdoc`	**`True`** - Always creates new ASICE container (even if signable file is already a ASICE container - ASICE in the ASICE container); **`False`** - If existing file is ASICE container, new signature will be added within existing ASICE container. If file is not ASICE, it will be added in ASICE container.

Example of single session signing

```
POST /api-sign/v1.0/CalculateDigest HTTP/1.1
Authorization: Bearer a477b3a3366768c07e4c458f518711b4b351e8d2c2f0f78a1524e4d3efd00603
Host: signapi-prep.eparaksts.lv
{
  "sessions": [
     {
        "sessionId": "fefdaec2b14bf2977d32a861fb49545244c654f7a4736dcc081ae1857a3a3dd4"
     }
  ],
  "certificate": "MIIG/j.......<sign certificate base64 here>.......xFP/IP==",
  "signAsPdf": false,
  "createNewEdoc": false
}
```

Example of batch session signing

```
POST /api-signing/v1.0/CalculateDigest HTTP/1.1
Authorization: Bearer a477b3a3366768c07e4c458f518711b4b351e8d2c2f0f78a1524e4d3efd00603
Host: signapi-prep.eparaksts.lv
{
  "sessions": [
          {
                  "sessionId": "fefdaec2b14bf2977d32a861fb49545244c654f7a4736dcc081ae1857a3a3dd4"
          },
          {
                  "sessionId": "fefdaec2b14bf2977d32a861fb49545244c654f7a4736dcc081ae18512121212"
          },
          {
                  "sessionId": "fefdaec2b14bf2977d32a861fb49545244c654f7a4736dcc081ae185bbbbbbbb"
          }
  ],
  "certificate": "MIIG/j.......<sign certificate base64 here>.......xFP/IP==",
  "signAsPdf": false,
  "createNewEdoc": false
}
```

Response

JSON object:

```
{
"sessionDigests":[
    {
      "sessionId": "string",
      "digest": "string",
      "error": {
          "code": "string",
          "message": "string"
      }
    }
  ],
  "digests_summary": "string",
  "algorithm": "string"
}
```

Property	Description
`sessionDigest`	Information about signable data session
`sessionDigest.sessionId`	File processing session identifier
`sessionDigest.diggest`	Signable data in *base64* format In case of server signing - received "**`diggest`**" property value shall be used (as is, without any reformatting) in Electronic signature provider API when Create a Digital Signature on the Server as "**`digest_value`**" property value. In case of signing with Smart Card by using LVRTC provided browser extension integration, then this value shall be converted to HEX.
`sessionDigests.error`	Session error if any
`sessionDigests.error.code`	Session error code
`sessionDigests.error.message`	Session error message
`digests_summary`	⚠️ This parameter must only be used to obtain the authorization from the end-user for generating a digital signature with a server signing identity enabled via password stored on the HSM. ⚠️ It is already precalculated digest summary. In case of server signing - "**`digests_summary`**" property value shall be used (as is, without any reformatting) in OAuth2.0 Authorization API when obtaining authorization code for signing operation as "**`digests_summary`**" property value.
`algorithm`	Algorithm of the digital signature

Example of single session signing

```
{
   "data": {
       "sessionDigests": [
           {
               "sessionId": "a37e460b4c65cb01a01dce5c58149806ca2d20dab22e99905d45128c4e693a90",
               "digest": "4xZX5G+R4gTbK2r6RlismZw4EBftvbSDcE3lXfpLMM4="
           }
       ],
       "digests_summary": "mnF3XVRWujh/Tsc3oA2HVGl0SI8VNb3pmscMcDhEzDo=",
       "algorithm": "SHA256"
   }
}
```

Example of batch session signing

```
{
  "data": {
          "sessionDigests": [
                 {
                    "sessionId": "fefdaec2b14bf2977d32a861fb49545244c654f7a4736dcc081ae1857a3a3dd4",
                    "digest": "wRX+DNmDdlDrMK8X/MEdersGZbsgTiSFHi26domxjwA="
                 },
                 {
                    "sessionId": "fefdaec2b14bf2977d32a861fb49545244c654f7a4736dcc081ae18512121212",
                    "digest": "wRX+DNmDdlDrMK8X/MEdersGZbsgTiSFHi26domxjwA="
                 },
                 {
                    "sessionId": "fefdaec2b14bf2977d32a861fb49545244c654f7a4736dcc081ae185bbbbbbbb",
                    "digest": "wRX+DNmDdlDrMK8X/MEdersGZbsgTiSFHi26domxjwA="
                 }
           ],
                  "digests_summary": "mnF3XVRWujh/Tsc3oA2HVGl0SI8VNb3pmscMcDhEzDo=",
         "algorithm": "SHA256"
  }
}
```

Paketes parakstšanas pabeigšanas piemrs

Scope

Ensure the finalization of the signature or seal (and container in case of ASICE).

Description

1. Attaches a signed data to the document;
2. Starts LVRTC timestamp request;
3. Attach timestamp and the revocation data (for example - OCSP) to the signature;
4. In case of successful execution the message "OK" is returned.

Request

The Service provider's application sends the following GET request using TLS:

```
POST /api-sign/v1.0/finalizeSigning
```

Authorization

The request must contain an *Authorization* header with an OAuth **Introspect access token** obtained via Integration Platform a Service provider's credentials grant flow.

Introspect piekuves talona (token) saemšana

Description

This operation obtains an *OAuth 2.0* access token. This operation can be invoked as part of an OAuth 2.0 Service provider's credentials grant flow.

Introspect access token

When the Service provider's credentials grant flow is used, the obtained access token demonstrates the administrative authorization of the S ervice provider's application making the call for accessing certain resources or services (i.e., without direct intervention of the resource's owner), or for accessing resources of the Service provider's application. Token is issued when the authorization server that processes the request is not associated to an identity provider. A token of this type can be used for accessing resources not associated to end-users or to end-user resources of any domain.

This type of access token is used to get access to Signature creation and validation service API's

Request

To obtain the token, the Service provider's application must send a request like the following to authorization server using TLS. This request is sent directly from the Service provider’s application to authorization server and does not go via the browser.

```
POST /trustedx-authserver/oauth/{as}/token
```

Parameter

Title	Type	Field	Description
as	path	mandatory	Use "lvrtc-eipsign-as"

Host:

Test environment: eidas-demo.eparaksts.lv

Production: eidas.eparaksts.lv

Content-Type Header

```
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
```

In HTTP POST request is necessary to incorporated the following main attribute: *Authorization* – API access token.

```
Authorization: Basic <API-Key>
```

How to generate API Access Key

Before Service provider access Integration platform API, LVRTC shall register Service provider as customer of Integration platform. After signing a contract with LVRTC (Test of Production environment) LVRTC generates Service Provider's application identifier – (*client_id*) and shared secret (*client_secret*), intended for the customer usage.
API Access Key (API Key) is generated from the Service provider's application identifier (*client_id*), a secret shared with the platform (*client_secret*) on the following basis:

Service provider's application identifier *client_id* are converted using the UTF-8 character encoding and URL encoding conditions.

For example, value "Portls" conversion result is "port%C4%81ls".

Service provider's application password *client_secret* is converted by using the UTF-8 character encoding and URL encoding conditions.

For example, value "drošba" conversion result is "dro%C5%A1%C4%ABba".

Both values of the previous two steps must be combined with separator colon ":" between them.

For example, by using previous examples, the result will be "port%C4%81ls:dro%C5%A1%C4%ABba".

Obtained value must be converted using base 64 encoding without line breaks.

For example, values "port%C4%81ls:dro%C5%A1%C4%ABba" conversion result is "CG94ydCVDNCUMWxzOmRybyVDNSVBMSVDNCVBQmJh".

"MIME Tools" tool in "Notepad + +" can be used for this purpose.

```
API-Key = base64[url_encode(utf8(<client_id>)) ':' url_encode(utf8(<client_secret>))]
```

Body

The content of the request for Introspect access token (used for access SignAPI service):

Property	Usage	Description
`grant_type`	mandatory	Must have the *client_credentials* value .
scope	mandatory	Must have the *urn:safelayer:eidas:oauth:token:introspect* value

Example (Introspect access token)

The following example shows a situation in which the Service provider's application with the identifier "*Portal*" and the password "*drošba*" authority shall transmit the request to the server with the identifier "*lvrtc-eips-as*":

```
POST /trustedx-authserver/oauth/lvrtc-eipsign-as/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic CG94ydCVDNCUMWxzOmRybyVDNSVBMSVDNCVBQmJh
Host: eidas-demo.eparaksts.lv
grant_type=client_credentials&
                        scope=urn%3Asafelayer%3Aeidas%3Aoauth%3Atoken%3Aintrospect
```

Response

In response, Integration platform authorization server issues a bearer-type OAuth 2.0 access token and returns it in a JSON structure.

```
{
"access_token" : {string},
"token_type" : "Bearer",
"expires_in" : {number}
}
```

Parameter

Property	Description
`access_token`	Access token generated by Authorization server. The token has the characteristics specified in the configuration of the authorization server that processed the request and consists of a random string of the number of bytes specified in the **Access token number of random bytes** (by default, 32), encoded in hexadecimal.
`token_type`	Type of access token. Always has the "*bearer*" value. (Bearer type OAuth 2.0 access token).
`expires_in`	Lifetime (in seconds) of the access token. The Service provider's application must perform the access the token authorizes before the token expires. This value can be configured in the **Token timeout** option of the authorization server (by default, 120 seconds). Once this timeout has expired, the token becomes invalid, and the Service provider's application must obtain another one if it wants to continue invoking the protected services.

Example

Introspect access token:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
{
 "scope": "urn:safelayer:eidas:oauth:token:introspect",
 "access_token": "dfffb0d7f90bed142464750cacad5e4b9e23f58ecb1d77e3bdf706ba208ad16a",
 "token_type": "Bearer",
 "expires_in": 600
}
```

Body

```
{
"sessionSignatureValues": [
    {
      "sessionId": "string",
      "signatureValue": "string"
    }
  ],
  "authCertificate": "string"
}
```

Property	Description
`sessionSignatureValues`	*Contains details of signed data*
`sessionSignatureValues.sessionId`	File processing session identifier. It is possible to specify multiple sessions.
`sessionSignatureValues.signatureValue`	Signed data in *base64* format. If there is specified multiple sessions, then each specified session shall contain its signed data.
`authCertificate`	End-user's authentication certificate

Examples

Example for single signature

```
POST /api-sign/v1.0/finalizeSigning HTTP/1.1
Authorization: Bearer a477b3a3366768c07e4c458f518711b4b351e8d2c2f0f78a1524e4d3efd00603
Host: signapi-prep.eparaksts.lv
{
  "sessionSignatureValues": [
     {
        "sessionId": "76fa04d8e5e2451b867af6ae667558395459fb9d082d31dadb9f22222f80a3d2",
        "signatureValue": "goU+aIw0sZHRwepWiooOdSb0eDbCZcBymXSgsLmYxCr6I6aZdDiG07vp4bbJMtGGPbTKEh6ZR+7eCmNfC02g
/hhlGU4OCr+LFhKFIcYqUW1VGvnEPx4eIJlw43pSIo6k
/It16RYyYWG1e2DVy1HuMoZEBScUAAN4tHFiHCiIsPuuzIYnRJtZMVi795dvgkEGxnghvzxp0rF+2eoT
/5gMKEcBxRIKKs2gWyFK+UIpiXigk0K8LTSEw4XXKDCMSVJ8Vp08nRm1grE0tjLY6tvUVs61Bh5ylC4d7Nh4gQB5VBJHBcng8lFAfXIokX0hD9eo
HBrx/bZ7uN1Co5U8H/HFvw=="
     }
  ],
  "authCertificate": "MIIGT....<auth cert base64 here>.......UoKew=="
}
```

Example for batch signing

```
POST /api-sign/v1.0/finalizeSigning HTTP/1.1
Authorization: Bearer a477b3a3366768c07e4c458f518711b4b351e8d2c2f0f78a1524e4d3efd00603
Host: signapi-prep.eparaksts.lv
{
  "sessionSignatureValues": [
    {
        "sessionId": "4bf47a3df5020ed537cc6337429db6dcbf7c299ab03474099a0b592190362d06",
        "signatureValue": "BJjtf/DjqBOTyFjjFci+HyzSnzhKE/diYpM9uf3HGLeX4VN+KYzwwgTiTnHn67zDMj5Qouu3ldBeIvswmD
/wDuwk+XyMaII7iVEg9CpvABhjDINXhtu5Wn/X5DLXQovN6Fd2xDBV1a392XfeBDtWNmWUr5P6ZZ7wfaCGuae1YyFUbfep6nL
/CC6iwKUmVMYyvHzr1ZHSpg/NAVvXC73Q3Bv3/XBYgPusuJZ5Sfv1Xv6PkyOMB9iC5307xk2V3+ptnDnKus+qYQioNhdKSmR4r92tQkZbtK
/kgPmkwpGO/B0wmI18ldzVS8gv7Fhorxp0sbDzjRpj0DI+UH6yEZn7ew=="
     },
     {
        "sessionId": "967e69ed16f7056dfe430f6292fd0f1dd9c28d87379bb7bcef94021ecc096642",
        "signatureValue": "ldH/38RET4fR4nLH5s6
/5uzkVbG4GPbsAsL0Wi9fAWWovXaR0ukzdhDnGVNuCQWfulOpEGKyTHvdfqbAKxrGlJ1QKcWvEluqk6bnYqUA
/vPOabPpXqWAvhpqf6R507aGCtbsUJ2OoNLyIl3BCP1fqV0uD12bz4x8Geqm2pjB+mj1rbp8kN4og5z4MrPd34Wi0n5JD+pFqd4
/UN5ttXB4Px5oZJGw99KvXU8sEDbyxn6qlojn7Z8OAQiwW6+x09degASXKGFHNbLBETbw8K/pUy1eaJDGQDn
/+pba7ygCMB4YnK2cl7c7hgvkzQjyYg3PQ5kpwCPiUj4Tt5Zd9u5o5A=="
     },
     {
        "sessionId": "3d87b13595af5094309028b7482df121988f97f4c04873ffd736a4186dd04069",
        "signatureValue": "fR3E287+05Gb8bPjqoiy0SGMtbKLGkopMb+AUgpS872bJ6qctEmpmZ5ZcsQ5j6HjmsZG34qg7Cvj
/AuviOEK8XPDrlFtGeHeq5Lb5Kjv7FJbDqoAP+DrRLmtt49/g9tSEbplZAiLiRCQg6uHpMr3gP7Jv6YWsQINArkAB7Gkpy
/0eA5egaabxmP78iz8Y74XGl2eBbkUmJCXyOPUTsPmkJZuzVMVIG5GFBoDz32AVUC0S2J5jOW8WUu/JPIeD/UqAE7Rs4c0jKz6PnjbZLFKRE
/qk2j2PCjzgd6kKNkNgPCA1mXXTPAd95Rq0FK25ZeHfZW93AaHDV1OFq85J6kMdA=="
     }
  ],
  "authCertificate": "MIIGT......<auth cert base64 here>........UoKew=="
}
```

Response

JSON object:

```
{
    "data": {
        "results": [
            {
                "sessionId": "string"
            }
        ]
    }
}
```

Property	Type	Description
sessionId	String (64)	File processing session identifier.

Example for single signature

```
{
    "data": {
        "results": [
            {
                "sessionId": "2ddec3e17a456417f48b044d71d2db2a31d00b62baea3de1211617e856f0f19d"
            }
        ]
    }
}
```

Example for batch signing


```
{
    "data": {
        "results": [
            {
                "sessionId": "4bf47a3df5020ed537cc6337429db6dcbf7c299ab03474099a0b592190362d06"
            },
            {
                "sessionId": "967e69ed16f7056dfe430f6292fd0f1dd9c28d87379bb7bcef94021ecc096642"
            },
            {
                "sessionId": "3d87b13595af5094309028b7482df121988f97f4c04873ffd736a4186dd04069"
            }
        ]
    }
}
```

Arhva laika zmoga pieprasjums

Scope

Provide a archive time stamp to already signed document.

Description

1. Requests time stamp with using client authentication certificate;
2. Received archive time stamp is added to signature with "ARCHIVE_TIMESTAMP" type;
3. In case of successful execution the session unique identifier is returned.

Request

The Service provider's application sends the following POST request using TLS:

```
POST /api-sign/v1.0/addArchive
```

Authorization

The request must contain an *Authorization* header with an OAuth **Introspect access token** obtained via Integration Platform a Service provider's credentials grant flow.

Introspect piekuves talona (token) saemšana

Description

This operation obtains an *OAuth 2.0* access token. This operation can be invoked as part of an OAuth 2.0 Service provider's credentials grant flow.

Introspect access token

When the Service provider's credentials grant flow is used, the obtained access token demonstrates the administrative authorization of the S ervice provider's application making the call for accessing certain resources or services (i.e., without direct intervention of the resource's owner), or for accessing resources of the Service provider's application. Token is issued when the authorization server that processes the request is not associated to an identity provider. A token of this type can be used for accessing resources not associated to end-users or to end-user resources of any domain.

This type of access token is used to get access to Signature creation and validation service API's

Request

To obtain the token, the Service provider's application must send a request like the following to authorization server using TLS. This request is sent directly from the Service provider’s application to authorization server and does not go via the browser.

```
POST /trustedx-authserver/oauth/{as}/token
```

Parameter

Title	Type	Field	Description
as	path	mandatory	Use "lvrtc-eipsign-as"

Host:

Test environment: eidas-demo.eparaksts.lv

Production: eidas.eparaksts.lv

Content-Type Header

```
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
```

In HTTP POST request is necessary to incorporated the following main attribute: *Authorization* – API access token.

```
Authorization: Basic <API-Key>
```

How to generate API Access Key

Before Service provider access Integration platform API, LVRTC shall register Service provider as customer of Integration platform. After signing a contract with LVRTC (Test of Production environment) LVRTC generates Service Provider's application identifier – (*client_id*) and shared secret (*client_secret*), intended for the customer usage.
API Access Key (API Key) is generated from the Service provider's application identifier (*client_id*), a secret shared with the platform (*client_secret*) on the following basis:

Service provider's application identifier *client_id* are converted using the UTF-8 character encoding and URL encoding conditions.

⚠ For example, value "Portls" conversion result is "port%C4%81ls".

Service provider's application password *client_secret* is converted by using the UTF-8 character encoding and URL encoding conditions.

⚠ For example, value "drošba" conversion result is "dro%C5%A1%C4%ABba".

Both values of the previous two steps must be combined with separator colon ":" between them.

⚠ For example, by using previous examples, the result will be "port%C4%81ls:dro%C5%A1%C4%ABba".

Obtained value must be converted using base 64 encoding without line breaks.

⚠ For example, values "port%C4%81ls:dro%C5%A1%C4%ABba" conversion result is "CG94ydCVDNCUMWxzOmRybyVDNSVBMSVDNCVBQmJh".

"MIME Tools" tool in "Notepad + +" can be used for this purpose.

```
API-Key = base64[url_encode(utf8(<client_id>)) ':' url_encode(utf8(<client_secret>))]
```

Body

The content of the request for Introspect access token (used for access SignAPI service):

Property	Usage	Description
`grant_type`	mandatory	Must have the *client_credentials* value .
scope	mandatory	Must have the *urn:safelayer:eidas:oauth:token:introspect* value

Example (Introspect access token)

The following example shows a situation in which the Service provider's application with the identifier "*Portal*" and the password "*drošba*" authority shall transmit the request to the server with the identifier "*lvrtc-eips-as*":

```
POST /trustedx-authserver/oauth/lvrtc-eipsign-as/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic CG94ydCVDNCUMWxzOmRybyVDNSVBMSVDNCVBQmJh
Host: eidas-demo.eparaksts.lv
grant_type=client_credentials&
                        scope=urn%3Asafelayer%3Aeidas%3Aoauth%3Atoken%3Aintrospect
```

Response

In response, Integration platform authorization server issues a bearer-type OAuth 2.0 access token and returns it in a JSON structure.

```
{
"access_token" : {string},
"token_type" : "Bearer",
"expires_in" : {number}
}
```

Parameter

Property	Description
`access_token`	Access token generated by Authorization server. The token has the characteristics specified in the configuration of the authorization server that processed the request and consists of a random string of the number of bytes specified in the **Access token number of random bytes** (by default, 32), encoded in hexadecimal.
`token_type`	Type of access token. Always has the "*bearer*" value. (Bearer type OAuth 2.0 access token).
`expires_in`	Lifetime (in seconds) of the access token. The Service provider's application must perform the access the token authorizes before the token expires. This value can be configured in the **Token timeout** option of the authorization server (by default, 120 seconds). Once this timeout has expired, the token becomes invalid, and the Service provider's application must obtain another one if it wants to continue invoking the protected services.

Example

Introspect access token:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
{
 "scope": "urn:safelayer:eidas:oauth:token:introspect",
 "access_token": "dfffb0d7f90bed142464750cacad5e4b9e23f58ecb1d77e3bdf706ba208ad16a",
 "token_type": "Bearer",
 "expires_in": 600
}
```

Body

```
{
  "sessions": [
    {
      "sessionId": "{String}"
    }
  ],
  "authCertificate": "{String}"
}
```

Property	Description
`sessionId`	File processing session identifier
`authCertificate`	End-user's authentication certificate

Example

```
POST /api-sign/v1.0/addArchive HTTP/1.1
Authorization: Bearer a477b3a3366768c07e4c458f518711b4b351e8d2c2f0f78a1524e4d3efd00603
Host: signapi-prep.eparaksts.lv
{
  "sessions": [
    {
      "sessionId": "26ae33853f7df73eaa84346a04a188db1614305aee43de0da667c67a4d371490"
    }
  ],
  "authCertificate":
"MIIGTjCCBDagAwIBAgIQGd6hCV2AEjVdTWDw5EEurDANBgkqhkiG9w0BAQsFADCBgzELMAkGA1UEBhMCTFYxOTA3BgNVBAoMMFZBUyBMYXR2aWphcyBWYWxzdHMgcmFkaW8gdW4gdGVsZXbEq3ppamFzIGNlbnRyczEaMBgGA1UEYQwRTlRSTFYtNDAwMDMwMTEyMDMxHTAbBgNVBAMMFERFTU8gTFYgZUlEIElDQSAyMDE3MB4XDTE5MDgwOTEyMDI1NloXDTIyMDgwOTEyMDI1NlowcDELMAkGA1UEBhMCTFYxHDAaBgNVBAMME0FORFJJUyBQQVJBVURaScWFxaAxFTATBgNVBAQMDFBBUkFVRFpJxYXFoDEPMA0GA1UEKgwGQU5EUklTMRswGQYDVQQFExJQTk9MVi0zMjEyMTUtNzkxNTkwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCS0tRy5CYE8Bz0yWmCIftJ0AIBHCvCW68AJPRmcJRNB0lCmXJoJvNKt9jnsgXLzUCgylK4hb5BmpbMP8Pt1TB2IIBNYIg/MdiwwAiAJi9OChCdJrlj0tpbZO3WPlTr3TjihsYxjvImCEwciPWXGV+Y5FJSnfnlMgZ22SMdiGRT5rrZ0v122+ULfVqMJc5s
/Fufws3vXuNRBewuzlCM6dcRmwl05qr0/Y7rPVR57kId+2dZD
/lWB0aXUE320Cr3u2J0y5iXS4zKUpNrrMozWXinVqhdpdF1l8BM6Kti99Kw1MyEADaRa8hNfyNEAoucjEj2OBuLEP1myII/Xnoj3
/yxAgMBAAGjggHOMIIByjAMBgNVHRMBAf8EAjAAMA4GA1UdDwEB
/wQEAwIHgDATBgNVHSUEDDAKBggrBgEFBQcDAjAdBgNVHQ4EFgQUppUtW5WAE82oSZPmnak+Y9394CQwHwYDVR0jBBgwFoAUj2jOvOLHQCFTCUK75Z4djEvNvTgwgYsGA1UdIASBgzCBgDA7BgYEAI96AQEwMTAvBggrBgEFBQcCARYjaHR0cHM6Ly93d3cuZXBhcmFrc3RzLmx2L3JlcG9zaXRvcnkwQQYMKwYBBAGB+j0CAQMBMDEwLwYIKwYBBQUHAgEWI2h0dHBzOi8vd3d3LmVwYXJha3N0cy5sdi9yZXBvc2l0b3J5MH0GCCsGAQUFBwEBBHEwbzBCBggrBgEFBQcwAoY2aHR0cDovL2RlbW8uZXBhcmFrc3RzLmx2L2NlcnQvZGVtb19MVl9lSURfSUNBXzIwMTcuY3J0MCkGCCsGAQUFBzABhh1odHRwOi8vb2NzcC5wcmVwLmVwYXJha3N0cy5sdjBIBgNVHR8EQTA
/MD2gO6A5hjdodHRwOi8vZGVtby5lcGFyYWtzdHMubHYvY3JsL2RlbW9fTFZfZUlEX0lDQV8yMDE3XzguY3JsMA0GCSqGSIb3DQEBCwUAA4ICAQBkgLwrpoAIx6FVE1NKdoTntzyQBi04+0uBnJdVOs6Zf9AHnZJQon88aorZEqPc0Y4D2
/DRQ58EhsEwULN8Us8zfdnd2QM6wpHsHTfzP0+moDLebRJQwzItsN+CiJxvziX7OVIXiS/mNL28mQL4mIW5bh4fbmx/34Dp6b+
/sTjaQUmTxyQUWI/FY8rQFiZs/Mp8B1PC6xbnUuYlcsiwesdGapG2WwGF1orVehMnpPQbwB3ZY6JkD
/vrrkqJnj8FwHRUYpswSDnmqZJPYTfiK5OoMc9yolH31r+m5h6DD3YkSnfKoxvrfRHQ8//+MWlWH+0W74ZPCNnRwKgAERmAL
/3fagWvpSnBPeWy+K/dhfGPyLKKLH/xJrU7FZ0VHTjOtbTIvXorVMX5Ab0aWqv+xOuemKPFD5nzMCBUudXJan0a22RfbWaLm0NUvb
/Oz+BN+NcRNc8wKs1Jr3asxNh5F7gxzqMHrxoK6zqCHRvUHPwlwYrGeu3j2vGgE5zinSQi9dBCSsFu7YEh2XZT1r7/4kMjolWX7wSafqO/Zuj
/15LNLmsaOIwgDoOfu0VL/WyjN6mWZyJ9RNG9uBYFJSX+jOpAU6g1eIU7cdKydPMCyTjvGSMvfGZ2/3J1ggvdQm7uO93PJyu9V
/D5raLXm4tPnIng9/VRc8SSdFcdks4PjUoKew=="
}
```

Response

JSON object:

```
{
    "data": {
        "results": [
            {
                "sessionId": {String}
            }
        ]
    }
}
```

Property	Description
`sessionId`	File processing session identifier

Example

```
{
    "data": {
        "results": [
            {
                "sessionId": "26ae33853f7df73eaa84346a04a188db1614305aee43de0da667c67a4d371490"
            }
        ]
    }
}
```

eZmoga izveidošana

Scope

Puts organization’s electronic seal on a file.

Description

Creates electronic seal signature using organization’s electronic seal certificate, timestamp using authentication certificate and the revocation data (for example - OCSP).

Request

The Service provider's application sends the following POST request using TLS:

```
POST /api-sign/v1.0/eSealCreate
```

Authorization

The request must contain an *Authorization* header with an OAuth **Introspect access token** obtained via Integration Platform a Service provider's credentials grant flow.

Introspect piekuves talona (token) saemšana

Description

This operation obtains an *OAuth 2.0* access token. This operation can be invoked as part of an OAuth 2.0 Service provider's credentials grant flow.

Introspect access token

When the Service provider's credentials grant flow is used, the obtained access token demonstrates the administrative authorization of the S ervice provider's application making the call for accessing certain resources or services (i.e., without direct intervention of the resource's owner), or for accessing resources of the Service provider's application. Token is issued when the authorization server that processes the request is not associated to an identity provider. A token of this type can be used for accessing resources not associated to end-users or to end-user resources of any domain.

⚠ This type of access token is used to get access to Signature creation and validation service API's

Request

To obtain the token, the Service provider's application must send a request like the following to authorization server using TLS. This request is sent directly from the Service provider’s application to authorization server and does not go via the browser.

```
POST /trustedx-authserver/oauth/{as}/token
```

Parameter

Title	Type	Field	Description
as	path	mandatory	Use "lvrtc-eipsign-as"

⚠ Host:

Test environment: eidas-demo.eparaksts.lv

Production: eidas.eparaksts.lv

Content-Type Header

```
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
```

In HTTP POST request is necessary to incorporated the following main attribute: *Authorization* – API access token.

```
Authorization: Basic <API-Key>
```

How to generate API Access Key

Before Service provider access Integration platform API, LVRTC shall register Service provider as customer of Integration platform. After signing a contract with LVRTC (Test of Production environment) LVRTC generates Service Provider's application identifier – (*client_id*) and shared secret (*client_secret*), intended for the customer usage.
API Access Key (API Key) is generated from the Service provider's application identifier (*client_id*), a secret shared with the platform (*client_secret*) on the following basis:

Service provider's application identifier *client_id* are converted using the UTF-8 character encoding and URL encoding conditions.

⚠ For example, value "Portls" conversion result is "port%C4%81ls".

Service provider's application password *client_secret* is converted by using the UTF-8 character encoding and URL encoding conditions.

⚠ For example, value "drošba" conversion result is "dro%C5%A1%C4%ABba".

Both values of the previous two steps must be combined with separator colon ":" between them.

⚠ For example, by using previous examples, the result will be "port%C4%81ls:dro%C5%A1%C4%ABba".

Obtained value must be converted using base 64 encoding without line breaks.

⚠ For example, values "port%C4%81ls:dro%C5%A1%C4%ABba" conversion result is "CG94ydCVDNCUMWxzOmRybyVDNSVBMSVDNCVBQmJh".

"MIME Tools" tool in "Notepad + +" can be used for this purpose.

```
API-Key = base64[url_encode(utf8(<client_id>)) ':' url_encode(utf8(<client_secret>))]
```

Body

The content of the request for Introspect access token (used for access SignAPI service):

Property	Usage	Description
`grant_type`	mandatory	Must have the *client_credentials* value .
scope	mandatory	Must have the *urn:safelayer:eidas:oauth:token:introspect* value

Example (Introspect access token)

The following example shows a situation in which the Service provider's application with the identifier "*Portal*" and the password "*drošba*" authority shall transmit the request to the server with the identifier "*lvrtc-eips-as*":

```
POST /trustedx-authserver/oauth/lvrtc-eipsign-as/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic CG94ydCVDNCUMWxzOmRybyVDNSVBMSVDNCVBQmJh
Host: eidas-demo.eparaksts.lv
grant_type=client_credentials&
                        scope=urn%3Asafelayer%3Aeidas%3Aoauth%3Atoken%3Aintrospect
```

Response

In response, Integration platform authorization server issues a bearer-type OAuth 2.0 access token and returns it in a JSON structure.

```
{
"access_token" : {string},
"token_type" : "Bearer",
"expires_in" : {number}
}
```

Parameter

Property	Description
`access_token`	Access token generated by Authorization server. The token has the characteristics specified in the configuration of the authorization server that processed the request and consists of a random string of the number of bytes specified in the **Access token number of random bytes** (by default, 32), encoded in hexadecimal.
`token_type`	Type of access token. Always has the "*bearer*" value. (Bearer type OAuth 2.0 access token).
`expires_in`	Lifetime (in seconds) of the access token. The Service provider's application must perform the access the token authorizes before the token expires. This value can be configured in the **Token timeout** option of the authorization server (by default, 120 seconds). Once this timeout has expired, the token becomes invalid, and the Service provider's application must obtain another one if it wants to continue invoking the protected services.

Example

Introspect access token:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
{
 "scope": "urn:safelayer:eidas:oauth:token:introspect",
 "access_token": "dfffb0d7f90bed142464750cacad5e4b9e23f58ecb1d77e3bdf706ba208ad16a",
 "token_type": "Bearer",
 "expires_in": 600
}
```

Body

```
{
  "sessions": [
    {
      "sessionId": "string"
    }
  ],
  "signAsPdf": false,
  "createNewEdoc": false,
  "signKey": "string",
  "signKeyPassword": "string",
  "authCertificate": "string"
}
```

Property	Description
`sessionId`	File processing session identifier *Multiple sessions can be sealed at once*
`signAsPdf`	**`True`** - will be signed as PDF **`False`** - will cerate XAdES signature in ASICE container (EDOC).
`createNewEdoc`	**`True`** - Always creates new ASICE container (even if signable file is already a ASICE container - ASICE in the ASICE container); **`False`** - If existing file is ASICE container, new signature will be added within existing ASICE container.
`signKey`	eSeal certificate key in PFX file format encoded in base64.
`signKeyPassword`	eSeal certificate key password encrypted with API central authentication certificate (issued by LVRTC) public key encoded in base64 ⚠ RSA Encryption with SHA1 padding
`authCertificate`	Authentication certificate related to eSeal for Timestamp request in PEM format

⚠ Key shall be provided in "*pfx*" format

Encription

```
byte[] signKeyPasswordBytes = Encoding.UTF8.GetBytes(req.SignKeyPassword);
byte[] signKeyPasswordBytesEncrypted = publicKey.Encrypt(signKeyPasswordBytes, RSAEncryptionPadding.Pkcs1);
req.SignKeyPassword = Convert.ToBase64String(signKeyPasswordBytesEncrypted);
```

Example

```
POST /api-sign/v1.0/eSealCreate  HTTP/1.1
Authorization: Bearer a477b3a3366768c07e4c458f518711b4b351e8d2c2f0f78a1524e4d3efd00603
Host: signapi-prep.eparaksts.lv
{
  "sessions": [
     {
        "sessionId": "702e9106e3fe2987d04a0bf7d585050477c1c0dce4d5b76fafc1178b537bc891"
     }
  ],
  "signAsPdf": false,
  "createNewEdoc": false,
  "signKey": "gbfb43564/70OeyYmqoLJ+YAU1kl+vwGVHT0L+ywky80EO8qx/kMZ7hq1Rm9p
/SjzexaS0lIH4yf0OJ74BdZCRGaeBfjZWHiQxBdQSXyeuCqDgawbcOjNMjQeLUK5s4a6T17WXjsPTqPulu8mdyhV+tPhwUE
/UdtOcSrJeY+ZkGMsn1i+YcNk1cg12+A/zQZ8lWach7M7Fj7gYhNaHiNW5JUREJKGgVKU4rSEE1T2LosJ9H
/b+I3Fj0AVR3Cw57W+VSxn8bXqQG8kq+MWdeomaprRvFQoSwY+MFUedqXgVNOioO8lARJNcd2duMLiy4PKEheqq6rZFftSlYulv+o9nPsbZXMO7H
",
  "signKeyPassword": "6hCV2AEjVdTWDw5EEurDANBgkqhkiG9w0BAQsFADCBgzELMAkGA1UEBhM",
   "authCertificate":
"MIIGTjCCBDagAwIBAgIQGd6hCV2AEjVdTWDw5EEurDANBgkqhkiG9w0BAQsFADCBgzELMAkGA1UEBhMCTFYxOTA3BgNVBAoMMFZBUyBMYXR2aWphcyBWYWxzdHMgcmFkaW8gdW4gdGVsZXbEq3ppamFzIGNlbnRyczEaMBgGA1UEYQwRTlRSTFYtNDAwMDMwMTEyMDMxHTAbBgNVBAMMFERFTU8gTFYgZUlEIElDQSAyMDE3MB4XDTE5MDgwOTEyMDI1NloXDTIyMDgwOTEyMDI1NlowcDELMAkGA1UEBhMCTFYxHDAaBgNVBAMME0FORFJJUyBQQVJBVURaScWFxaAxFTATBgNVBAQMDFBBUkFVRFpJxYXFoDEPMA0GA1UEKgwGQU5EUklTMRswGQYDVQQFExJQTk9MVi0zMjEyMTUtNzkxNTkwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCS0tRy5CYE8Bz0yWmCIftJ0AIBHCvCW68AJPRmcJRNB0lCmXJoJvNKt9jnsgXLzUCgylK4hb5BmpbMP8Pt1TB2IIBNYIg/MdiwwAiAJi9OChCdJrlj0tpbZO3WPlTr3TjihsYxjvImCEwciPWXGV+Y5FJSnfnlMgZ22SMdiGRT5rrZ0v122+ULfVqMJc5s
/Fufws3vXuNRBewuzlCM6dcRmwl05qr0/Y7rPVR57kId+2dZD
/lWB0aXUE320Cr3u2J0y5iXS4zKUpNrrMozWXinVqhdpdF1l8BM6Kti99Kw1MyEADaRa8hNfyNEAoucjEj2OBuLEP1myII/Xnoj3
/yxAgMBAAGjggHOMIIByjAMBgNVHRMBAf8EAjAAMA4GA1UdDwEB
/wQEAwIHgDATBgNVHSUEDDAKBggrBgEFBQcDAjAdBgNVHQ4EFgQUppUtW5WAE82oSZPmnak+Y9394CQwHwYDVR0jBBgwFoAUj2jOvOLHQCFTCUK75Z4djEvNvTgwgYsGA1UdIASBgzCBgDA7BgYEAI96AQEwMTAvBggrBgEFBQcCARYjaHR0cHM6Ly93d3cuZXBhcmFrc3RzLmx2L3JlcG9zaXRvcnkwQQYMKwYBBAGB+j0CAQMBMDEwLwYIKwYBBQUHAgEWI2h0dHBzOi8vd3d3LmVwYXJha3N0cy5sdi9yZXBvc2l0b3J5MH0GCCsGAQUFBwEBBHEwbzBCBggrBgEFBQcwAoY2aHR0cDovL2RlbW8uZXBhcmFrc3RzLmx2L2NlcnQvZGVtb19MVl9lSURfSUNBXzIwMTcuY3J0MCkGCCsGAQUFBzABhh1odHRwOi8vb2NzcC5wcmVwLmVwYXJha3N0cy5sdjBIBgNVHR8EQTA
/MD2gO6A5hjdodHRwOi8vZGVtby5lcGFyYWtzdHMubHYvY3JsL2RlbW9fTFZfZUlEX0lDQV8yMDE3XzguY3JsMA0GCSqGSIb3DQEBCwUAA4ICAQBkgLwrpoAIx6FVE1NKdoTntzyQBi04+0uBnJdVOs6Zf9AHnZJQon88aorZEqPc0Y4D2
/DRQ58EhsEwULN8Us8zfdnd2QM6wpHsHTfzP0+moDLebRJQwzItsN+CiJxvziX7OVIXiS/mNL28mQL4mIW5bh4fbmx/34Dp6b+
/sTjaQUmTxyQUWI/FY8rQFiZs/Mp8B1PC6xbnUuYlcsiwesdGapG2WwGF1orVehMnpPQbwB3ZY6JkD
/vrrkqJnj8FwHRUYpswSDnmqZJPYTfiK5OoMc9yolH31r+m5h6DD3YkSnfKoxvrfRHQ8//+MWlWH+0W74ZPCNnRwKgAERmAL
/3fagWvpSnBPeWy+K/dhfGPyLKKLH/xJrU7FZ0VHTjOtbTIvXorVMX5Ab0aWqv+xOuemKPFD5nzMCBUudXJan0a22RfbWaLm0NUvb
/Oz+BN+NcRNc8wKs1Jr3asxNh5F7gxzqMHrxoK6zqCHRvUHPwlwYrGeu3j2vGgE5zinSQi9dBCSsFu7YEh2XZT1r7/4kMjolWX7wSafqO/Zuj
/15LNLmsaOIwgDoOfu0VL/WyjN6mWZyJ9RNG9uBYFJSX+jOpAU6g1eIU7cdKydPMCyTjvGSMvfGZ2/3J1ggvdQm7uO93PJyu9V
/D5raLXm4tPnIng9/VRc8SSdFcdks4PjUoKew=="}
```

Response

JSON object:

```
{
    "results": "string"[
        {
            "sessionId": "string",
            "error": {
                "code": "string",
                "message": "string"
            }
        }
    ]
}
```

Property	Description
`results`	Signing results: success if error == null or omitted
`results.sessionId`	File processing session identifier

`results.error`	Session error if any
`results.error.code`	Session error code
`results.error.message`	Session error message

Example


```
{
  "data": {
         "results": [
                {
                   "sessionId": "76fa04d8e5e2451b867af6ae667558395459fb9d082d31dadb9f22222f80a3d2"
                }
         ]
  }
}
```