|  |  |
| --- | --- |
|  | BOSA eSealing SOLUTION  Technical information |

## Summary

This document is intended for an FPS (Federal Public Service) that wishes to integrate the BOSA esealing solution ‘Federal Trust Services’ (FTS). It allows client applications sign a document with Private keys held on HSMs. This document, for example a PDF or XML, has been prepared or created by the FPS.

## Content

[Summary 2](#_Toc126241126)

[Content 2](#_Toc126241127)

[1. Introduction FTS esealing 3](#_Toc126241128)

[1.1. Sealing flow 3](#_Toc126241129)

[2. Getting started 5](#_Toc126241130)

[2.1. Environments and development process 5](#_Toc126241131)

[2.2. Code repositories 6](#_Toc126241132)

[2.3. Main URLs 7](#_Toc126241133)

[2.5. Signing profiles 8](#_Toc126241134)

[2.7. Example code 9](#_Toc126241135)

[3. Details 10](#_Toc126241136)

[3.1. The “getDataToSign” call 10](#_Toc126241137)

[3.2 signDocument 12](#_Toc126241138)

[3.5. Error codes 14](#_Toc126241139)

[4. Sample/test FPS code and service 15](#_Toc126241140)

|  |  |
| --- | --- |
|  |  |

## 

## Introduction FTS esealing

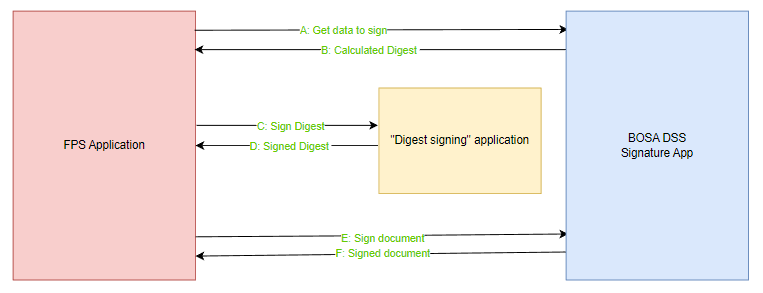
### 1.1. Sealing flow

Sealing is “automated sealing” or “machine sealing”, where there is no human decision to sign the document outside of the “design phase” decision to sign.

The FTS signing service help client applications:

* Adhere to the latest EIDAS regulations
* Take into account specific constraints of the Belgian Government PKI and their evolutions
* Assure quality of the signing process

The sealing is a three step flow:



1. **Calculate the Document Digest (/Hash)** 
   1. **Mandatory inputs of this operation:**
      1. **The document to sign**
      2. **The signing certificate**
      3. **The certificate chain of the signing certificate**
      4. **The requested signing profile**
   2. **The reply will contain:**
      1. **The calculated Digest**
      2. **The Digest algorithm**
      3. **The Signature date**
2. **Sign the Digest**
   1. **Typical inputs of this operation:**
      1. **The calculated Digest**
      2. **The Digest algorithm**
   2. “Digest signing” applications can require extra mandatory fields
3. Sign the Document
   1. **Mandatory inputs of this operation:**
      1. **The document to sign**
      2. **The signing certificate**
      3. **The certificate chain of the signing certificate**
      4. **The requested signing profile**
      5. **The calculated Digest**
      6. **The Digest algorithm**
      7. **The Signature date**
   2. **The reply is a signed file or an error**

## Getting started

### 2.1. Environments and development process

|  |  |  |
| --- | --- | --- |
| TA | **TEST** | continuously deployed from commits to “developer” code branch. |
| QA | **QUALITY** | continuously deployed from commits to “master” code branch. |
| INT | **INTEGRATION** | deployed from stable release. |
| PROD | **PRODUCTION** | deployed from stable release. IP filtering will be required. |

#### Day-to-Day development

* Development is done on “developer” code branch.
* Changes are deployed automatically to TA.
* New features are developed here.
* Bug fixing is done here.

#### Feature complete development

* Completed features are pushed to the “master” code branch.
* Changes are deployed automatically to QA.
* Development is done on TA, QA receives the changes when feature complete.
* Stable releases are created from QA: a consistent set of features is tagged on QA, it can
* then be deployed to INT for testing and to PROD when integration testing proves to be successful.

#### Deliverables and artifacts

* Development is done in an open source development model.
* The service is based on Docker containers.
* The BOSA service is based on a stable version that passed integration testing.

### 2.2. Code repositories

#### Code repositories

BOSA uses 2 code repositories.

**GitLab: internal**

* BOSA hosted and private
* Gitlab is the primary repository where BOSA development is done
* It has 2 code branches: “develop” and “master”
* The repository is integrated with the build & deploy pipeline that is used to host the service for all environments, which is why it is private
* The TA environment tracks the “develop” branch at any time
* The QA environment tracks the “master” branch at any time
* The INT and PROD environments are deployed using static/stable versions of the service that are tagged on the master branch

**GitHub: public**

* Public code repository
* Automated mirror of the BOSA GitLab “master” branch
* Stable release tags are mirrored as well: the version that is available on PROD can be checked out from the code
* Repositories
  + <https://github.com/Fedict/fts-mintest>

Code for mintest code example, a basic implementation how to call the service including several base scenario’s

* + <https://github.com/Fedict/fts-documentation>

Repository for documentation and architecture documents

* + <https://github.com/Fedict/fts-sign-validation>

Signing and validation engine

### 2.3. Main URLs

URLs are based on QA.

* Replace QA with TA or INT for those environments
* Remove QA in the URL for PROD

Example:

* *PROD: https://dp.fts.bosa.belgium.be*
* *TA : https://dp.ta.fts.bosa.belgium.be*
* *QA : https://dp.qa.fts.bosa.belgium.be  
  INT : https://dp.int.fts.bosa.belgium.be*

#### URLs

* <http://validate.ta.fts.bosa.belgium.be/signandvalidation/signing/getDataToSign>
  + URL to calculate the Digest
  + API endpoint
* <http://validate.ta.fts.bosa.belgium.be/signandvalidation/signing/signDocument>
  + URL to Sign the document
  + API endpoint

### 2.5. Signing profiles

To instruct the service what level of signature should be generated on a document, base profiles have been defined for common use cases. The profile definitions are available on <https://github.com/Fedict/fts-sign-validation/tree/master/parameters/signature>.

The current profiles that are readily available are:

|  |  |  |  |
| --- | --- | --- | --- |
| **PROFILENAME** | **PROFILE** | **FORMAT** | **REMARK** |
| JADES\_B | JadES\_BASELINE\_B | ENVELOPING | COMPACT\_SERIALIZATION |
| JADES\_T | JadES\_BASELINE\_T | ENVELOPING | JSON\_SERIALIZATION |
| JADES\_LT | JadES\_BASELINE\_LT | ENVELOPING | FLATTENED\_JSON\_SERIALIZATION |
| JADES\_LTA | JadES\_BASELINE\_LTA | ENVELOPING | JSON\_SERIALIZATION |
| PADES\_1 | PadES\_BASELINE\_B | ENVELOPED |  |
| PADES\_LTA | PadES\_BASELINE\_LTA | ENVELOPED |  |
| PADES\_LTA\_EXP\_ALLOW | PadES\_BASELINE\_LTA | ENVELOPED | **ALLOWS SIGNING WITH EXPIRED CERTIFICATES. NOT AVAILABLE IN INT/PROD** |
| XADES\_1 | XadES\_BASELINE\_B | ENVELOPED |  |
| XADES\_2 | XadES\_BASELINE\_B | ENVELOPING |  |
| XADES\_LT | XadES\_BASELINE\_LT | ENVELOPED |  |
| XADES\_LTA | XadES\_BASELINE\_LTA | ENVELOPED |  |
| MDOC\_XADES\_LTA | XadES\_BASELINE\_LTA | INTERNALLY DETACHED | **Custom Xades multi document for Justice** |
| XADES\_LTA\_EXP\_ALLOW | XadES\_BASELINE\_LTA | ENVELOPED | **ALLOWS SIGNING WITH EXPIRED CERTIFICATES. NOT AVAILABLE IN INT/PROD** |

For reference the resulting signatures include the following elements per international standards:

|  |
| --- |
| **B: SIGNATURE WITH CERTIFICATE CHAIN** |
| **T: B + TIME STAMP** |
| **LT: T + REVOCATION DATA** |
| **LTA: LT + 2ND TIME STAMP** |

While the B and LTA options should cover customer needs, T and/or LT profiles can easily be added.

### 2.7. Example code

BOSA DT maintains a code example how to integrate with the service for eSignatures that covers the common use cases. The source code is publicly available on <https://github.com/Fedict/fts-test-environment/tree/master/mintest> The example itself is deployed and hosted on all environments (in PROD it is only accessible from internal network) and can be accessed and tested using the following URL (example URL is for QA): <https://mintest.qa.fts.bosa.belgium.be>.

### **3.** **Details**

### 3.1. The “getDataToSign” call

This is a REST call (a HTTP POST containing a json string) from the FPS to the BOSA DSS server.

This operation will calculate the Dighest/Hash that will be signed later.

example request:

|  |
| --- |
| {  “clientSignatureParameters": {  "signingCertificate": {"encodedCertificate":"<Base64(X509 Certificate)>”},  "certificateChain": [  {"encodedCertificate": “<Base64(X509 Intermediate Certificate)>”},  {"encodedCertificate": “<Base64(X509 Root Certificate)>”}  ]  },  "signingProfileId":" PADES\_LTA",  "toSignDocument":{ "bytes":"<Base64(Raw document bytes)>" }  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **JSON Field** | **Mandatory** | **Values** | **Description** |
| signingCertificate | Yes | MIIEjTCCBBKgAwIBAgI.... | The Signing certificate (X509 / Base 64 encoded) |
| certificateChain | Yes |  | The chained array of “Issuing certificates” of the Signing certificate |
| certificateChain*.*encodedCertificate | Yes | MIIEjTCCBBKgAwIBAgI.... | An “Issuing certificate” (X509 / Base 64 encoded) |
| signingProfileId | Yes | XADES\_1 / PADES\_LTA | The signature profile requested. It is linked to the “in” file type. BOSA provides a list of the available values and their meaning. |
| toSignDocument.bytes | Yes | JVBERi0xLjQKJcOkw7zDtsOfCjIgMCBv.... | The file to sign (Base 64 encoded) |

example reply:

|  |
| --- |
| {  "digestAlgorithm" : "SHA256",  "digest" : "1+Ij9/JqR7biNUOrz4bNJXxCoO5djQdkJlhUhkswRJ8=",  "signingDate" : "2023-02-02T10:01:22.381+0000"  } |

|  |  |  |
| --- | --- | --- |
| **JSON Field** | **Values** | **Description** |
| digestAlgorithm | SHA224, SHA256, SHA384, SHA512, SHA3\_224, SHA3\_256, SHA3\_384, SHA3\_512, RIPEMD160, WHIRLPOOL…… | The algorithm to use when signing the Digest  **This algorithm is determined based on the Signing certificate, profile and the type of document to be signed** |
| Digest | 1+Ij9/JqR7biNUOrz4bNJXxCoO5djQdkJlhUhkswRJ8= | The digest (Base 64 encoded) |
| signingDate | 2023-02-02T10:01:22.381+0000 | The date that will be recorded as the signing date |

### 3.2 Sign the Digest / Hash

The API to call will depend on the “signing hash” service selected.

### 3.3 signDocument

This is a REST call (a HTTP POST containing a json string) from the FPS to the BOSA DSS server

example request:

|  |
| --- |
| {  "toSignDocument":{"bytes":"JVBERi0xLjQKJcOkw...."},  "signingProfileId":"PADES\_1",  "clientSignatureParameters":{  "signingCertificate":{"encodedCertificate":"MIIEjTCCBBKgAwIBAgIQS7...."},  "certificateChain":[  {"encodedCertificate":"MIIDSTCCAtCgA..."},  {"encodedCertificate":"MIICCDCCAY+gAw..."}  ],  "signingDate":"2023-02-02T10:01:22.381+0000"},  "signatureValue":"MGUCMA9PfLWY...."  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **JSON Field** | **Mandatory** | **Values** | **Description** |
| signingCertificate | Yes | MIIEjTCCBBKgAwIBAgI.... | The Signing certificate (X509 / Base 64 encoded)  MUST be the same as “getDataToSign” |
| certificateChain | Yes |  | The chained array of “Issuing certificates” of the Signing certificate  MUST be the same as “getDataToSign” |
| certificateChain -> encodedCertificate | Yes | MIIEjTCCBBKgAwIBAgI.... | An “Issuing certificate” (X509 / Base 64 encoded)  MUST be the same as “getDataToSign” |
| signingProfileId | Yes | XADES\_1 / PADES\_LTA | The signature profile requested. It is linked to the “in” file type. BOSA provides a list of the available values and their meaning.  MUST be the same as “getDataToSign” |
| toSignDocument -> bytes | Yes | JVBERi0xLjQKJcOkw7zDtsOfCjIgMCBv.... | The file to sign (Base 64 encoded)  MUST be the same as “getDataToSign” |
| signingDate | Yes | 2023-02-02T10:01:22.381+0000 | MUST come from the reply of “getDataToSign” |
| signatureValue | Yes | MGUCMA9PfLWY.... | The output of the SignDigest call |

example reply:

|  |
| --- |
| {  "bytes" : "JVBERi0xLjQKJcOkw7zDtsO....",  "name" : "Filename.pdf"  } |

|  |  |  |
| --- | --- | --- |
| **JSON Field** | **Values** | **Description** |
| *bytes* | JVBERi0xLjQKJcOkw7zDtsO.... | The bytes of the signed file (Base 64 encoded) |
| *name* | File.xml | The name of the output file. Not mandatory to use this name |

### 3.5. Error codes

URL: <https://github.com/Fedict/fts-documentation/tree/master/FPS_info> (file: error\_constants.txt).

Error codes that the service might include on callback to the relying party.

|  |  |  |
| --- | --- | --- |
| Error constant | Explanation | Origin |
| NO\_DOC\_TO\_VALIDATE | DSSDocument is null | Backend |
| NO\_CERT\_TO\_VALIDATE | The certificate is missing | Backend |
| CERT\_REVOKED | Certificate (probably the signing cert) revoked | Backend |
| INTERNAL\_ERR | Unexpected error occured | Backend |
| INVALID\_DOC | Document validation (after signing) failed | Backend |
| UNKNOWN\_PROFILE | Unknown signature profile | Backend |
| EMPTY\_PARAM | Empty (null) parameter in request | Backend |
| INVALID\_SIGNATURE\_LEVEL | Invalid signature level for document | Backend |
| SIGN\_PERIOD\_EXPIRED | The “SignDocumentForToken” call was executed too late (See “signTimeout”) | Backend |
| \_ERROR | Suffix added to generic backend errors | Backend |
| INVALID\_PARAM | Returned by the GetToken calls when input is invalid | Backend |