

# Specification of Standard Audit File for Tax (JPK) service interfaces

Information Technology Center of the Ministry of Finance

23 November 2023

Version 4.1

| Date       | Version | Description  |
|------------|---------|--|
| 23.05.2016 | 1.3     | Publication of technical specification of the Standard Audit File for Tax services.  |
| 10.06.2016 | 1.4     | <ol style="list-style-type: none"> <li>1. Change of packed file splitting method from TAR to SPLIT binary file splitting method.</li> <li>2. Status method: <ul style="list-style-type: none"> <li>• change of returned content for http code: 200 and 400.</li> </ul> </li> <li>3. InitUploadSigned method for http code: 200 <ul style="list-style-type: none"> <li>• change of type for TimeoutInSec properties from Timespan to int</li> </ul> </li> <li>4. Change of XSD scheme of metadata file: <ul style="list-style-type: none"> <li>• adding the JPKAH (JPK ad hoc) document type for files uploaded under the audit,</li> <li>• validation of EncryptionKey name to EncryptionKey (misspelling),</li> <li>• validation of REST API version format,</li> <li>• validation of file name format,</li> <li>• adding total number of parts of split file and sequence number for individual parts,</li> <li>• deletion of type and mode attributes from the list of partial files FileSignatureList,</li> <li>• adding the (Packaging) element to the list of partial files FileSignatureList with the option of selecting the type of file split and compression. The existing options include zip compression (deflate) with binary splitting - SplitZip element with type (split) and mode (zip) elements,</li> <li>• adding the Encryption element to the list of partial files FileSignatureList with the option of encryption algorithm AES256 - AES element with size (256), block (16), mode (CBC) and padding (PKCS#7) attributes and IV (Initialization Vector) element with bytes (16) and encoding (Base64) attributes.</li> </ul> </li> </ol> |
| 17.06.2016 | 1.5     | <p>Change of XSD scheme of metadata file:</p> <ul style="list-style-type: none"> <li>• specification of the supported version of REST API - 01.02.01.20160617,</li> <li>• change of FileName regular expression,</li> <li>• padding the response code set for the Status method.</li> </ul>  |
| 04.07.2016 | 1.6     | <ol style="list-style-type: none"> <li>1. Adding the description for encryption key ciphering.</li> <li>2. Change to the interface description – translating messages into Polish.</li> <li>3. Adding (RequestId) the request identifier to the response structure for http code: 400 and 500.</li> <li>4. Extending the error response code set (400 Bad Request) of the InitUploadSigned method.</li> </ol>  |

| Date       | Version | Description  |
|------------|---------|--|
|            |         | <ol style="list-style-type: none"> <li>Adding information on permissible transformations for metadata signature.</li> <li>Limiting of hash function value length in the XSD scheme of metadata file.</li> </ol>  |
| 20.07.2016 | 1.7     | <ol style="list-style-type: none"> <li>Extending the error response code set (400 Bad Request) of the InitUploadSigned method.</li> <li>Adding the examples of valid session initiation responses with the InitUploadSigned method.</li> <li>Placing the examples of using SDK software tools of the Put Blob method.</li> <li>Adding information on the parameter enabling verification of signature with qualified certified when initiating the session with the InitUploadSigned method in test environment.</li> </ol>  |
| 29.07.2016 | 2.0     | Specification of Zip compression scheme.   |
| 30.09.2016 | 2.1     | <ol style="list-style-type: none"> <li>Padding the the error response code set (400 Bad Request) of the InitUploadSigned method.</li> <li>Specification of domain addresses used in Azure Storage space.</li> </ol>  |
| 31.01.2017 | 2.2     | Change of examples of valid session initiation responses with the InitUploadSigned method.   |
| 31.03.2017 | 2.3     | <ol style="list-style-type: none"> <li>Extending the description of metadata signature functionalities with support for the European qualified signature and Trusted Profile signature.</li> <li>Extending the description of domain addresses used in Azure Storage space.</li> </ol>   |
| 11.05.2020 | 3.0     | <ol style="list-style-type: none"> <li>Extending the description of preparation of authentication metadata with the option of using authorization data (authorization with personal data and amount values from the previous settlements).</li> <li>Updating the status codes: <ul style="list-style-type: none"> <li>adding the new code 136 returned in the InitUploadSigned method,</li> <li>deletion of absent codes (102, 110, 301, 302, 303, 403, 404, 409, 411, 414),</li> <li>adding the new codes 417, 418, 419, 420, 422, 423, 424 returned in the Status method.</li> </ul> </li> <li>Adding the description of powers of attorney.</li> <li>Extending the scope of cloud storage facilities (p. 2.2).</li> </ol> |
| 25.09.2020 | 3.1     | Updating the status codes:   |

| Date                                   | Version | Description   |
|--|---------|---|
|  |         | <ul style="list-style-type: none"> <li>adding the new code 155 returned in the InitUploadSigned method.</li> </ul>  |
| 06.11.2020                             | 3.2     | Updating the status codes: <ul style="list-style-type: none"> <li>adding the new error code 411 returned in the Status method.</li> </ul>   |
| 21.01.2021                             | 3.3     | Adding the CUK(1) file support.   |
| 27.05.2021                             | 3.4     | 1. Adding the CUK(2) and ALK(1) files support.<br>2. Updating the status codes: <ul style="list-style-type: none"> <li>adding the new error codes 99 and 101 returned in the InitUploadSigned method,</li> <li>adding the new codes 425 and 426 returned in the Status method.</li> </ul> |
| 13.01.2022<br>07.12.2022<br>16.01.2023 | 3.5     | 3. Adding the JPK_V7M(2) and JPK_V7K(2) files support.<br>4. Adding the ITP (1) and ITP-Z (1) files support.<br>5. Adding the ITP (2) and ITP-Z (2) files support.  |
| 21.06.2023                             | 3.6     | Adding the JPK_GV(1) file support.  |
| 13.10.2023                             | 4.0     | 1. Adding the CESOP: PSP-IP (4) support.<br>2. „Block scheme of data uploading preparation steps” – adding an optional step with document signing.  |
| 23.11.2023                             | 4.1     | 1. Adding the CESOP: PSP-FR (1) support.<br>2. Editorial changes to the document.   |

## Table of contents

|        |  |    |
|--------|--|----|
| 1      | Preparation of JPK data .....  | 6  |
| 1.1    | File format and document type.....   | 6  |
| 1.2    | Preparation of JPK documents .....   | 6  |
| 1.1.1. | Data compression .....   | 8  |
| 1.1.2. | Data encryption.....   | 8  |
| 1.1.3. | Encryption key ciphering.....  | 8  |
| 1.3    | Preparation of authentication metadata .....                                 | 8  |
| 1.3.1  | Qualified or trusted signature .....   | 9  |
| 1.3.2  | Authorization data .....   | 9  |
| 1.4    | Document type .....  | 10 |
| 2      | Specification of interface accepting the JPK documents for the clients ..... | 10 |
| 2.1    | Introduction .....   | 10 |
| 2.2    | Interface description .....  | 10 |
| 2.2.1  | InitUploadSigned .....   | 11 |
| 2.2.2  | Put Blob.....  | 22 |
| 2.2.3  | FinishUpload .....   | 24 |
| 2.2.4  | Status.....  | 26 |

# 1 Preparation of JPK data

## 1.1 File format and document type

The file format is always .xml. When an XML document is mentioned, it is understood to mean as a type of folded document, i.e. the value of the "DocumentType" field. It should be noted that the XML file does not have to be an XML document.

## 1.2 Preparation of JPK documents

The Standard Audit File for Tax (JPK) data will be prepared by the client (e.g. in the ERP system) in the form of XML files compatible with the XSD scheme published by:

- Ministry of Finance at <https://epuap.gov.pl/wps/portal/strefa-urzednika/inne-systemy/crwde> or on the official website of the JPK Structures – Ministry of Finance – National Revenue Administration - Portal Gov.pl ([www.gov.pl](http://www.gov.pl)).
- European Commission at: [https://taxation-customs.ec.europa.eu/taxation-1/central-electronic-system-payment-information-cesop\\_en](https://taxation-customs.ec.europa.eu/taxation-1/central-electronic-system-payment-information-cesop_en).

Names of schemes published in the Central Repository of Electronic Document Models of the Electronic Platform of Public Administration Services (ePUAP):

- **JPK\_V7M(1), JPK\_V7M(2)** MONTHLY STATEMENT AND RECORD FOR VALUE ADDED TAX (IN THE FORM OF STANDARD AUDIT FILE FOR TAX).
- **JPK\_V7K(1), JPK\_V7K(2)** QUARTERLY STATEMENT AND RECORD FOR VALUE ADDED TAX (IN THE FORM OF STANDARD AUDIT FILE FOR TAX).
- **CUK (1), CUK (2)** INFORMATION ON FOODSTUFFS FEE.
- **ALK (1)** INFORMATION ON FEE FOR PERMIT FOR WHOLESALE TRADING IN ALCOHOLIC BEVERAGES OF UP TO 300 ML IN VOLUME.
- **ITP (1), ITP (2), ITP-Z (1), ITP-Z (2)** INFORMATION ON PAYMENT TRANSACTIONS USING PAYMENT TERMINALS.
- **JPK\_GV (1)** INTERNAL RECORD OF VAT GROUP MEMBERS.

In addition to the above-mentioned diagrams published in CRWDE e-PUAP, the diagrams published on the BIP MF/KAS website are also supported:

- **JPK\_FA(4)** VAT INVOICE (IN THE FORM OF STANDARD AUDIT FILE FOR TAX),
- **JPK\_FA\_RR(1)** VAT INVOICE FLAT-RATE FARMERS (IN THE FORM OF STANDARD AUDIT FILE FOR TAX),
- **JPK\_EWP(3)** REVENUE RECORDS (IN THE FORM OF STANDARD AUDIT FILE FOR TAX (3)),
- **JPK\_EWP(2)** REVENUE RECORDS (IN THE FORM OF STANDARD AUDIT FILE FOR TAX (2)),
- **JPK\_EWP(1)** REVENUE RECORDS (IN THE FORM OF STANDARD AUDIT FILE FOR TAX (1)),
- **JPK\_PKPIR(2)** TAX REVENUE AND EXPENSE LEDGER (IN THE FORM OF STANDARD AUDIT FILE FOR TAX (2)),
- **JPK\_KR(1)** ACCOUNTING BOOKS (IN THE FORM OF STANDARD AUDIT FILE FOR TAX (1)),

- **JPk\_MAG(1)** WAREHOUSE (IN THE FORM OF STANDARD AUDIT FILE FOR TAX (1)),
- **JPk\_WB(1)** BANK STATEMENT (IN THE FORM OF STANDARD AUDIT FILE FOR TAX (1)),
- **PSP-FR(1)** REGISTRATION FORM FOR PAYMENT INSTITUTIONS OBLIGED TO REPORT UNDER THE CENTRAL ELECTRONIC SYSTEM OF PAYMENT INFORMATION,
- **PSP-IP(4)** REPORT FROM PAYMENT INSTITUTIONS OBLIGED TO REPORT UNDER THE CENTRAL ELECTRONIC SYSTEM OF PAYMENT INFORMATION.

Each document described with a valid scheme should constitute a separate XML file. The generated XML file should be UTF-8 encoded. The JPK documents are prepared for uploading in line with the scheme presented below:

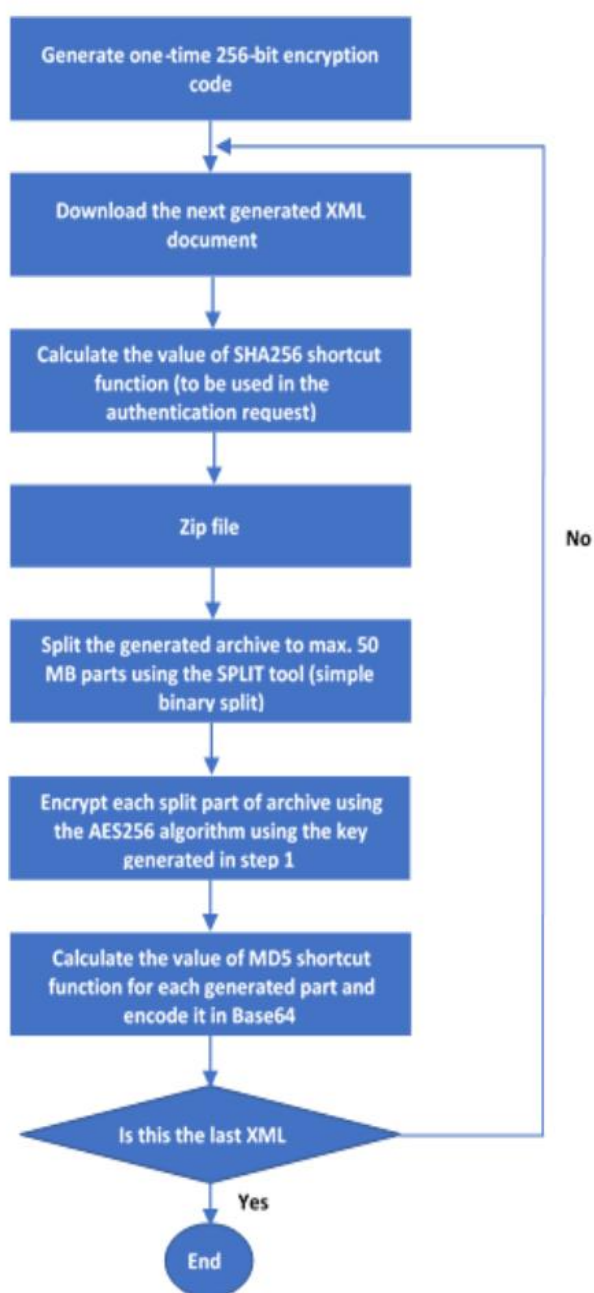


Fig. 1 Block scheme of data uploading preparation steps

#### 1.1.1. Data compression

The generated document will be compressed to file in ZIP format and subject to binary split to parts not exceeding 60 MB in size.

The required compression method is the ZIP file format using the DEFLATE algorithm without the splitting option (split/multipart). The compression should result in a single ZIP file containing a single document. If the size of resulting ZIP file exceeds 60MB, it should be subject to binary split to relevant number of 60MB parts and the last part of size not exceeding 60MB.

Using this approach enables the use of commonly available tools and ensures easy deployment on various platforms.

#### 1.1.2. Data encryption

The next stage after file zipping is their encryption. Files are encrypted with the use of AES256 algorithm with key generated by the client.

##### AES algorithm specification:

|                       |                             |
|-----------------------|-----------------------------|
| Key Size              | 256 bits / 32 bytes         |
| Cipher Mode           | CBC (Cipher Block Chaining) |
| Padding               | PKCS#7                      |
| Block Size            | 16 bytes                    |
| Initialization Vector | 16 bytes                    |

##### Encryption procedure:

- **Key generation:** the client generates a random 256-bit key.
- **Archive encryption:** all segments of compressed archive (see point 1.1) are encrypted using the abovementioned AES256 algorithm and the generated key.
- **Key encryption:** the key used for file encryption is then ciphered using the RSA asymmetric algorithm. This is made with the use of public key certificate made available by the Ministry of Finance.
- **Adding key to metadata:** Upon encrypting, the key is added to the metadata file described further in the documentation.

#### 1.1.3. Encryption key ciphering

Encryption key should be ciphered with the RSA asymmetric algorithm with the use of public key certificate made available by the Ministry of Finance. While deploying the encryption scheme, the following RSA algorithm specification should be used:

|             |                           |
|-------------|---------------------------|
| Key Size    | 256 bits / 32 bytes       |
| Cipher Mode | ECB (Electronic Codebook) |
| Padding     | PKCS#1                    |
| Block Size  | 256 bytes                 |

### 1.3 Preparation of authentication metadata

Upon preparation of the essential documents compatible with the relevant file type scheme, the client wishing to upload data must prepare the authentication data in the form of a dedicated XML uploaded in the InitUploadSigned method (described in the next chapter).

The metadata file must be authenticated using one of the following techniques:



1. using:
  - a. qualified signature (Polish or European),
  - b. trusted signature
2. embedding the AuthData element containing the encrypted authorization data.

### 1.3.1 Qualified or trusted signature

The metadata file must be signed digitally with the **Polish or European qualified signature** or **trusted signature** in line with the XAdES Basic Electronic Signature algorithm in the form of XML file in accordance with the <http://www.w3.org/2000/09/xmldsig> scheme i.e. XAdES-BES in **Enveloped** version (signature as an additional ds:Signature element in the original XML) or **Enveloping** (original document embedded as an element in the signed structure). When signing, the signed object can be transferred using the <http://www.w3.org/2000/09/xmldsig#base64> encoding.

The hash function used for signature purposes should be RSA-SHA256.

The example of authentication metadata is available later in the document, which discusses the InitUploadSigned method adopting the authentication metadata.

### 1.3.2 Authorization data

When using the amount authorization method, the AuthData element should be padded:

```
<xs:element name="AuthData" minOccurs="0" maxOccurs="1">
```

```
<xs:annotation>
```

**<xs:documentation>**This optional field should contain the XML document compatible with the published SIG-2008\_v2-0.xsd schema encrypted with the use of AES256 symmetric algorithm. The same key, which is used to encipher the part of the zipped JPK archive and enclosed to this metadata file, should be used. Encrypted data encoding algorithm is Base64.**</xs:documentation>**

```
</xs:annotation>
```

```
<xs:simpleType>
```

```
<xs:restriction base="xs:string"/>
```

```
</xs:simpleType>
```

```
</xs:element>
```

This field should contain the XML document compatible with the published SIG-2008\_v2-0.xsd scheme encrypted with the use of AES256 symmetric algorithm (generated by the client), **The same key**, which is used to encipher the part of the zipped JPK archive and enclosed to this metadata file, should be used. Encrypted data encoding algorithm is Base64.

#### Authorization data encryption parameters:

|                       |                             |
|-----------------------|-----------------------------|
| Key Size              | 256 bits / 32 bytes         |
| Cipher Mode           | CBC (Cipher Block Chaining) |
| Padding               | PKCS#7                      |
| Block Size            | 16 bytes                    |
| Initialization Vector | 16 bytes                    |

## 1.4 Document type

Depending on the type of uploaded file, it must have the appropriate document type embedded in the scheme. The following document types are available:

1. JPK for JPK, CUK, ALK and ITP files
2. JPKAH for JPK files on request
3. XML for PSP files

The document type is embedded as DocumentType, exemplary use:

“<DocumentType>JPK</DocumentType>”

## 2 Specification of interface accepting the JPK documents for the clients

### 2.1 Introduction

The document acceptance system uses the RESTful system operating via HTTPS protocol.

### 2.2 Interface description

The essential part of interface for the ERP clients is structured from the following methods:

- InitUploadSigned
- Put Blob
- FinishUpload
- Status

**Deployment of test environment available at:**

<https://test-e-dokumenty.mf.gov.pl/>

The addresses of individual methods are as follows:

<https://test-e-dokumenty.mf.gov.pl/api/Storage/InitUploadSigned>

<https://test-e-dokumenty.mf.gov.pl/api/Storage/Status/{referenceNumber}>

<https://test-e-dokumenty.mf.gov.pl/api/Storage/FinishUpload>

Addresses of cloud storage facilities to which the JPK files are uploaded:

<https://taxdocumentstorage00tst.blob.core.windows.net>

<https://taxdocumentstorage01tst.blob.core.windows.net>

<https://taxdocumentstorage02tst.blob.core.windows.net>

<https://taxdocumentstorage97tst.blob.core.windows.net>

<https://taxdocumentstorage98tst.blob.core.windows.net>

<https://taxdocumentstorage99tst.blob.core.windows.net>

**Deployment of production environment is available at:**

<https://e-dokumenty.mf.gov.pl/>

Addresses of individual methods are as follows:

<https://e-dokumenty.mf.gov.pl/api/Storage/InitUploadSigned>

<https://e-dokumenty.mf.gov.pl/api/Storage/Status/{referenceNumber}>

<https://e-dokumenty.mf.gov.pl/api/Storage/FinishUpload>

Addresses of cloud storage facilities to which the JPK files are uploaded:

<https://taxdocumentstorage00.blob.core.windows.net>

<https://taxdocumentstorage01.blob.core.windows.net>

<https://taxdocumentstorage02.blob.core.windows.net>

<https://taxdocumentstorage97.blob.core.windows.net>

<https://taxdocumentstorage98.blob.core.windows.net>

<https://taxdocumentstorage99.blob.core.windows.net>

used domain names are verifiable using the regular expression:

**`https://[0-9]{2}taxdocumentstorage[0-9]{2}.blob.core.windows.net/[0-9]{2}`**

Detailed description of the methods operation is presented below.

### 2.2.1 InitUploadSigned

The client session initiation method. Its call is a precondition to upload data using the Put Blob method of the Azure service.

| Name                  | InitUploadSigned |
|-----------------------|------------------|
| Method type           | POST             |
| Uploaded content type | application/xml  |
| Returned content type | application/json |
| Maximum request size  | 100KB            |

Description of parameters provided in the method address:

| Name                                    | Description   | Type | Validation  |
|---|---|------|---|
| <b>enableValidateQualifiedSignature</b> | If <b>true</b> value is transmitted ( <b>in test environment</b> ) the system shall verify whether the signed file was signed with valid Polish or European qualified signature or trusted signature. | bool | Optional – permissible values: <b>true</b> , <b>false</b> |

The method address with enabled verification of qualified signature:

<https://test-e-dokumenty.mf.gov.pl/api/Storage/InitUploadSigned?enableValidateQualifiedSignature=true>

Description of the XML structure being the message body:

| Name                           | Description   | Type                                       | Validation   |
|--------------------------------|---|--|--|
| <b>InitUpload</b>              | Metadata for the InitUpload method                      | Object                                     | Required   |
| <b>DocumentType</b>            | Name of uploaded document type.                         | String                                     | Required – permissible values:<br><b>JPK</b> – XML documents compatible with schema issued by the Ministry of Finance, upload on a regular basis<br><b>JPKAH</b> – XML documents compatible with schema issued by the Ministry of Finance and uploaded on request under the audit<br><b>XML</b> – XML documents compatible with schema issued by the entities other than the Ministry of Finance (for <b>PSP-IP(4)</b> ) |
| <b>Version</b>                 | REST API version to which the request is addressed      | String                                     | Required,<br>01.02.01.20160617<br>Required,<br>01.03.01.20231001<br>(for <b>PSP-IP(4)</b> )  |
| <b>EncryptionKey</b>           | Symmetric key encrypted with asymmetric algorithm (RSA) | String                                     | Required   |
| <b>EncryptionKey.algorithm</b> | Algorithm used to encrypt the symmetric key             | String – permissible values:<br><b>RSA</b> | Required   |

| Name                          | Description   | Type  | Validation   |
|-------------------------------|---|---|--|
| <b>EncryptionKey.mode</b>     | Encryption mode   | String – permissible values:<br><b>ECB</b>    | Required   |
| <b>EncryptionKey.padding</b>  | Encryption key padding format                                   | String – permissible values:<br><b>PKCS#1</b> | Required   |
| <b>EncryptionKey.encoding</b> | Key value encryption algorithm                                  | String – permissible values:<br><b>Base64</b> | Required   |
| <b>DocumentList</b>           | List of uploaded documents                                      | List of Document type objects                 | Required. The list must include at least one document.                               |
| <b>Document</b>               | Metadata of uploaded document                                   | Object  | Required   |
| <b>FormCode</b>               | FormCode embedded in the XML file heading                       | String  | Required   |
| <b>FormCode.systemCode</b>    | systemCode attribute of the FormCode element of the XML file    | String  | Required   |
| <b>FormCode.schemaVersion</b> | schemaVersion attribute of the FormCode element of the XML file | String  | Required   |
| <b>FileName</b>               | JPk file name   | String  | Required, unique, format: [a-zA-Z0-9_\.\\-]{5,55} for example JPK_VAT_2016-07-01.xml |
| <b>ContentLength</b>          | Total document size   | Long  | Required   |
| <b>HashValue</b>              | Hash of the entire document                                     | String  | Required   |

| Name                                 | Description  | Type   | Validation   |
|--------------------------------------|--|--|--|
| <b>HashValue.algorithm</b>           | Name of hash function algorithm  | String – permissible values:<br><b>SHA-256</b> | Required   |
| <b>HashValue.encoding</b>            | Encoding algorithm of the hash function value  | String – permissible values:<br><b>Base64</b>  | Required   |
| <b>FileSignatureList</b>             | Metadata of files contained in the document. If the size of uploaded file is below 60MB, the list includes only one file | List of FileSignature type objects             | Required. The list must include at least one element |
| <b>FileSignatureList.filesNumber</b> | Number of all file parts   | int  | Required   |
| <b>Packaging</b>                     | Possible types of document splitting and compression   | Selection list                                 | Required   |
| <b>SplitZip</b>                      | Type of document splitting and compression   | Object   | Required   |
| <b>SplitZip.type</b>                 | Type of method to split a documents to parts   | String – permissible values:<br><b>split</b>   | Required   |
| <b>SplitZip.mode</b>                 | Type of compression algorithm  | String – permissible values:<br><b>zip</b>     | Required   |
| <b>Encryption</b>                    | Possible partial file encryption methods   | Selection list                                 | Required   |
| <b>AES</b>                           | Partial file encryption method   | Object   | Required   |
| <b>AES.size</b>                      | Size of encryption key expressed in bits   | Int – permissible values:<br><b>256</b>        | Required   |

| Name                 | Description                                       | Type  | Validation   |
|----------------------|---|---|--|
| <b>AES.block</b>     | Size of encryption block expressed in bytes       | Int – permissible values:<br><b>16</b>        | Required   |
| <b>AES.mode</b>      | Encryption mode                                   | String – permissible values:<br><b>CBC</b>    | Required   |
| <b>AES.padding</b>   | Encryption block padding method                   | String – permissible values:<br><b>PKCS#7</b> | Required   |
| <b>IV</b>            | Initialization vector of the encryption algorithm | String  | Required   |
| <b>IV.bytes</b>      | Size of initialization vector in bytes            | String – permissible values:<br><b>16</b>     | Required   |
| <b>IV.encoding</b>   | Initialization vector value encoding method       | String – permissible values:<br><b>Base64</b> | Required   |
| <b>FileSignature</b> | File metadata                                     | Object  | Required   |
| <b>OrdinalNumber</b> | Ordinal number of the next part                   | Int   | Required, unique   |
| <b>FileName</b>      | Name of file uploaded to Azure Storage.           | String  | Required, unique, format: [a-zA-Z0-9_\.\\-]{5,55} for example JPK_VAT_2016-07-01.xml.zip.001.aes |
| <b>ContentLength</b> | Length of file uploaded to Azure Storage          | Int   | Required.<br>Maximum size is   |

| Name                       | Description  | Type  | Validation                      |
|----------------------------|--|---|---------------------------------|
|                            |  |   | 62914560 bytes<br>(60MB)        |
| <b>HashValue</b>           | The hash function value of the file uploaded to Azure Storage, encoded in Base64 (do not convert to hex before conversion to Base64)   | String  | Required. Length: 24 characters |
| <b>HashValue.algorithm</b> | Name of hash function algorithm,   | String – permissible values:<br><b>MD5</b>    | Required                        |
| <b>HashValue.encoding</b>  | Encoding algorithm of hash function value  | String – permissible values:<br><b>Base64</b> | Required                        |
| <b>AuthData</b>            | This optional field should contain the XML file compatible with the published SIG-2008_v2-0.xsd scheme encrypted with the use of AES256 symmetric algorithm. The same key, which is used to encipher the part of the zipped JPK archive and enclosed to this metadata file, should be used. Encrypted data encoding algorithm is Base64. | String  | Optional                        |

Hash value of file uploaded to Storage (**HashValue** element in **FileSignatureType** type) is the value of has function in line with MDS encoded using Base64.

The XSD scheme of XML document being the request content is made available at <https://www.podatki.gov.pl/jednolity-plik-kontrolny/> in the “JPK\_VAT z deklaracją” (*JPK\_VAT with statement*) section. This location includes the exemplary metadata signed in the XAdES-BES format with a non-qualified (self-signed) signature.

Valid version of the XSD scheme for the CESOP project is available at:  
<https://www.gov.pl/web/kas/dostawcy-uslug-platniczych>



The **InitUploadSigned** method returns three types of responses:

| Response code             | Description                             |
|---------------------------|---|
| <b>200 – OK</b>           | Session initiated successfully          |
| <b>400 – Bad Request</b>  | Invalid request. Erroneous service call |
| <b>500 – Server Error</b> | Erroneous request processing            |

Description of JSON structure (**application/json**) of valid response (**200 – OK**):

| Name                           | Description  | Type                                     |
|--------------------------------|--|--|
| <b>ReferenceNumber</b>         | Initiated session identifier   | String                                   |
| <b>TimeoutInSec</b>            | Lifetime (in seconds) of authentication key used to upload documents (depends on the number of files declared for uploading)   | Int                                      |
| <b>RequestToUploadFileList</b> | List of metadata used to create the request to upload file to Azure Storage  | List of RequestToUploadFile type objects |
| <b>RequestToUploadFile</b>     | Metadata used to create the request to upload file to Azure Storage  | Object                                   |
| <b>BlobName</b>                | Name of blob to which the file will be saved   | String                                   |
| <b>FileName</b>                | Name of file   | String                                   |
| <b>Url</b>                     | Address, to which the file will be uploaded using the <i>Put Blob</i> method. The address is generated on a dynamic basis and its scheme may change.                         | String                                   |
| <b>Method</b>                  | The method to send the <i>Put Blob</i> request   | String                                   |
| <b>HeaderList</b>              | List of headers required to create the <i>Put Blob</i> request.<br><br>The returned headers are generated on a dynamic basis. Their names and number of elements may change. | List of keys and values                  |
| <b>Key</b>                     | Header key   | String                                   |
| <b>Value</b>                   | Header value   | String                                   |

Exemplary content of valid response (200 - OK):

```
{
  "ReferenceNumber": "d4fd41850323d2f6000000b013016327",
  "TimeoutInSec": 900,
```

"RequestToUploadFileList": [

{

"BlobName": "8377ed3d-1b05-4c76-b718-6fddd46fd298",

"FileName": "jpk\_vat\_100-01.xml.zip.aes",

"Url":

"https://taxdocumentstorage09tst.blob.core.windows.net/d4fd41850323d2f6000000b013016327/8377ed3d-1b05-4c76-b718-6fddd46fd298?sv=2015-07-

08&sr=b&si=d4fd41850323d2f6000000b013016327&sig=yFXyJdsPPkbEOiQwVs5ccLEYEU0lxQHldbVvPfPciXw%3D",

"Method": "PUT",

"HeaderList": [

{

"Key": "Content-MD5",

"Value": "eXkPLHMM+dHB5GCFoeAvsA=="

},

{

"Key": "x-ms-blob-type",

"Value": "BlockBlob"

}

]

},

{

"BlobName": "0a80a089-bc10-41e1-a74d-70fd45f27aa3",

"FileName": "jpk\_vat\_100-02.xml.zip.aes",

"Url":

"https://taxdocumentstorage09tst.blob.core.windows.net/d4fd41850323d2f6000000b013016327/0a80a089-bc10-41e1-a74d-70fd45f27aa3?sv=2015-07-

08&sr=b&si=d4fd41850323d2f6000000b013016327&sig=Fj%2BGjn7hCKIM6hSvMBGWBxSOyV7V%2FLMM9pnenbaoxks%3D",

"Method": "PUT",

"HeaderList": [

{

"Key": "Content-MD5",

"Value": "NZew85QTb16mFLzx9cyKzA=="

},

{

```

    "Key": "x-ms-blob-type",
    "Value": "BlockBlob"
  }
]
}
]
}

```

Response for the exemplary file signed with non-qualified signature in XAdES- BES format (enveloping) published on the website in the JPK-VAT-TEST-0001.ZIP archive:

```

{
  "ReferenceNumber": " ef7d17780087346e0000004c0c7982ec",
  "TimeoutInSec": 900,
  "RequestToUploadFileList": [
    {
      "BlobName": "094951bc-ba54-404e-b2c8-df2591ad0e17",
      "FileName": "JPK-VAT-TEST-0001.xml.zip.aes",
      "Url":
        "https://taxdocumentstorage03tst.blob.core.windows.net/ef7d17780087346e0000004c0c7982ec/094951bc-ba54-404e-b2c8-df2591ad0e17?sv=2015-07-08&sr=b&si=ef7d17780087346e0000004c0c7982ec&sig=kN7LlprYkIP9uxod%2F1gcaDGN8WjbEbfDIA4GXuuzOmk%3D",
      "Method": "PUT",
      "HeaderList": [
        { "Key": "Content-MD5", "Value": "5YnivEH4gz5Wg5E8M2XwAQ==" },
        { "Key": "x-ms-blob-type", "Value": "BlockBlob" }
      ]
    }
  ]
}

```

Response for the exemplary file signed with non-qualified signature in XAdES- BES (enveloped) format, published on the website in the JPK-VAT-TEST-0000.ZIP archive:

```

{
  "ReferenceNumber": " ef81ecf9011a546c0000004d72be8011",
  "TimeoutInSec": 900,
  "RequestToUploadFileList": [

```

```
{
  "BlobName": "55a19799-5f1d-4336-9051-197dc53e5adf",
  "FileName": "JPK-VAT-TEST-0001.xml.zip.aes",
  "Url":
    "https://taxdocumentstorage02tst.blob.core.windows.net/ef81ecf9011a546c0000004d72be8011/55a19799-5f1d-4336-9051-197dc53e5adf?sv=2015-07-08&sr=b&si=ef81ecf9011a546c0000004d72be8011&sig=HeLYQd8RfRucs4KGgWxITEU36OgQuqSe1RUXZ10n8%2Bs%3D",
  "Method": "PUT",
  "HeaderList": [
    { "Key": "Content-MD5", "Value": "5YnivEH4gz5Wg5E8M2XwAQ==" },
    { "Key": "x-ms-blob-type", "Value": "BlockBlob" }
  ]
}
```

#### Description of JSON structure (application/json) of response (400 – Bad Request):

| Name             | Description                   | Type        |
|------------------|-------------------------------|-------------|
| <b>Message</b>   | Error message                 | String      |
| <b>Code</b>      | Error code                    | String      |
| <b>Errors</b>    | Optionally. Error table       | String list |
| <b>RequestId</b> | Unique bad request identifier | GUID        |

#### Specification of codes embedded in response (400 – Bad Request):

| Code       | Message  | Description  |
|------------|--|--|
| <b>99</b>  | Invalid character encoding in the xml file             | The provided document is not encoded in UTF-8 format   |
| <b>100</b> | Invalid XML  | The provided document is not the XML document  |
| <b>101</b> | Invalid character encoding declaration in the xml file | The provided document has invalid character encoding declaration (other than <code>&lt;?xml version="1.0" encoding="utf-8"?&gt;</code> ) |
| <b>110</b> | Document not signed                                    | The provided document is not signed as required by the specification   |
| <b>111</b> | Signature in the other format than XAdES-BES           |  |

| Code | Message   | Description  |
|------|---|--|
| 112  | Invalid signature. Verification impossible.   | Unexpected error occurred during signature verification.   |
| 113  | Signature in non-supported external (detached) format   | The supported signature formats are enveloped and enveloping   |
| 114  | Difficulties with reading the signed object   |  |
| 120  | Signature verified negatively   | Positive verification of signature failed.   |
| 130  | Signature references verified negatively. Data probably modified.                             |  |
| 135  | Document with non-qualified signature   | Authenticity of qualified signature verified in the production environment.                              |
| 136  | Document contains both electronic signature and authorization data                            | The document may be authenticated using only one technique   |
| 140  | Uploaded file incompatible with XSD scheme  | Verification document using the InitUpload.xsd scheme failed   |
| 150  | Non-supported form code:<br>"specific systemCode"   | Non-supported form code  |
| 155  | Uploaded file is invalid. At least two partial files of the same hash declared.               | The error consists in declaration in the initupload file of at least two partial files of the same hash. |
| 160  | "Specific HashValue" value is not encoded in Base64   | Hash of files declared for uploading must be encoded in Base64.  |
| 170  | Duplicate of processed document uploaded. Reference number of the original document: XXXXXXXX | Duplicates are verified on the basis of the SHA-256 hash value of the declared JPK document              |

Exemplary response:

```
{
  "Message": "Signature verified negatively",
  "Code": 120,
  "RequestId": "172dc3cc-5b97-48de-91dd-6903587cba19"
}
```

**Description of JSON structure (application/json) of response (500 – Internal Server Error):**

| Name      | Description                   | Type   |
|-----------|-------------------------------|--------|
| Message   | Error message                 | String |
| RequestId | Unique bad request identifier | GUID   |

Exemplary response:

```
{
  "Message": "Internal system error ",
  "RequestId": "172dc3cc-5b97-48de-91dd-6903587cba19"
}
```

### 2.2.2 Put Blob

The method uploading the essential JPK documents. Directly deployed by the storage space service by Azure (Azure Storage).

Its complete documentation is available at:

<https://learn.microsoft.com/en-us/rest/api/storageservices/Put-Blob>

Uploading via http client

#### Request address:

`https://<storage_account_name>.blob.core.windows.net/<reference_number>/<blob_name>`

Complete address at which the client uploads the JPK documents is returned by the `InitUploadSigned` method. The Shared Access Signature i.e. the one-time key enabling the client to place documents in the dedicated container is a part of the returned address. The SAS key is generated on a one-time basis and valid for a predefined time and in the predefined part of space of Azure Storage – thus provides high security level.

#### Request method:

Returned by `InitUploadSigned`.

#### Request header

Returned by `InitUploadSigned`. Used request headers:

| Request header        | Description  |
|-----------------------|--|
| <i>x-ms-blob-type</i> | Required. Specifies the type of blob. Permissible value is <b>BlockBlob</b> .  |
| <i>Content-MD5</i>    | Optional. MD5 has function value. This hash is used to verify data integrity during transfer. While using this value, Azure Storage automatically verifies the hash value of data received with the declared ones. If both values differ, the procedure fails with error code 400 (Bad Request). |

#### Request content

The request content contains the uploaded file.

Complete documentation on request headers – and other details of interaction with Azure Storage – is available at already provided address:

<https://msdn.microsoft.com/en-us/library/azure/dd179451.aspx>

**The Put Blob method returns the following responses:**

| Response code        | Description                                |
|----------------------|--|
| <b>201 – Created</b> | File successfully uploaded in Azure space. |
| <b>4xx</b>           | Erroneous service call                     |
| <b>5xx</b>           | Erroneous request processing               |

Response (201 – Created):

Empty response body

**Responses 4xx and 5xx return the error message in XML form (application/xml):**

| Name           | Description            | Type   |
|----------------|------------------------|--------|
| <b>Error</b>   | Key structural element | Object |
| <b>Code</b>    | Descriptive error code | String |
| <b>Message</b> | Error message          | String |

Example:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<Error>
```

```
  <Code>AuthenticationFailed</Code>
```

```
  <Message>Server failed to authenticate the request. Make sure the value of Authorization header  
is formed correctly including the signature.
```

```
RequestId:a5124e1c-0001-0056-06b3-ddc62c000000 Time:2016-07-  
14T09:40:13.7833645Z</Message>
```

```
  <AuthenticationErrorDetail>SAS identifier cannot be found for specified signed  
identifier</AuthenticationErrorDetail>
```

```
</Error>
```

### Uploading via SDK

Available deployments: .NET, Node.js, Java, C++, PHP, Ruby, Python, iOS, Xamarin.

<https://azure.microsoft.com/en-gb/documentation/articles/storage-dotnet-how-to-use-blobs/>

Example:

### Message returned by InitUploadSigned:

```
{
  "ReferenceNumber": "d8cb2f0f014381ab000000b012f8a3d6",
  "TimeoutInSec": 900,
  "RequestToUploadFileList": [
    {
      "BlobName": "b42748d3-0660-4d81-afc2-3c250fbcdbeef",
```

```
"FileName": "jpk_vat_100.xml.zip.aes",
"Url":
"https://taxdocumentstorage10tst.blob.core.windows.net/d8cb2f0f014381ab000000b012f8a3d6/b4
2748d3-0660-4d81-afc2-3c250fbcdbe?sv=2015-07-
08&sr=b&si=d8cb2f0f014381ab000000b012f8a3d6&sig=2y%2BZ3cjcyBbBnCM6Mw9a4EPN2KA%2B0
1kgf9fro%2FK6Xgw%3D",
"Method": "PUT",
"HeaderList": [
{ "Key": "Content-MD5", "Value": "eXkPLHMM+dHB5GCFoeAvsA==" },
{ "Key": "x-ms-blob-type", "Value": "BlockBlob" }
]
}
]
```

#### Uploading file in.NET:

```
var absoluteUri =
"https://taxdocumentstorage10tst.blob.core.windows.net/d8cb2f0f014381ab000000b012f8a3d6/b4
2748d3-0660-4d81-afc2-3c250fbcdbe";

var sas = "sv=2015-07-
08&sr=b&si=d8cb2f0f014381ab000000b012f8a3d6&sig=2y%2BZ3cjcyBbBnCM6Mw9a4EPN2KA%2B0
1kgf9fro%2FK6Xgw%3D";

var blob = new CloudBlockBlob(new Uri(absoluteUri), new StorageCredentials(sas)); using (var
stream = new FileStream("jpk_vat_100-01.xml.zip.aes", FileMode.Open))
{
blob.UploadFromStream(stream);
}
```

### 2.2.3 FinishUpload

This method ends the session. Its call is a precondition for successful completion of the uploading procedure. It verifies the required files, using the name and MD5 of the values declared in InitUploadSigned. Failure to call is equivalent to the recognition that the session is interrupted.

| Name                  | FinishUpload     |
|-----------------------|------------------|
| Method type           | POST             |
| Uploaded content type | application/json |
| Returned content type | application/json |
| Maximum request size  | 100KB            |



**Description of JSON (application/json) structure being the message body:**

| Name                     | Description                                       | Type        | Validation   |
|--------------------------|---|-------------|--|
| <b>ReferenceNumber</b>   | Session identifier                                | String      | Required   |
| <b>AzureBlobNameList</b> | List of names of blobs contained in Azure Storage | String list | Required. The list must contain the same number of elements that was uploaded to Azure Storage |

Example:

```
{
  "ReferenceNumber": "e8505c4703e5fd5b000000b04bc6f43f"
  "AzureBlobNameList": [
    "d1eadd0e-ccd5-44ab-85e7-2f2a552e7f17",
    "5c3ceb5f-8c5d-4720-9005-7c7d1d88f121"
  ],
}
```

**The FinishUpload method returns three types of responses:**

| Response code             | Description                             |
|---------------------------|---|
| <b>200 – OK</b>           | Session completed successfully          |
| <b>400 – Bad Request</b>  | Invalid request. Erroneous service call |
| <b>500 – Server Error</b> | Erroneous request processing            |

Response (200 – OK):

Empty response body

**Description of JSON (application/json) response (400 – Bad Request):**

| Name             | Description                   | Type        |
|------------------|-------------------------------|-------------|
| <b>Message</b>   | Error message                 | String      |
| <b>Errors</b>    | Optionally. Error table       | String list |
| <b>RequestId</b> | Unique bad request identifier | GUID        |

Example:

```
{
```

```
"Message": "Bad request"
```

```
"Errors": "[Reference number is required]"
```

```
"RequestId": "172dc3cc-5b97-48de-91dd-6903587cba19"
```

```
}
```

**Description of JSON (application/json) response (500 – Internal Server Error):**

| Name      | Description             | Type   |
|-----------|-------------------------|--------|
| Message   | Error message           | String |
| RequestId | Optionally. Error table | GUID   |

Example:

```
{
```

```
"Message": "Internal server error",
```

```
"RequestId": "172dc3cc-5b97-48de-91dd-6903587cba19"
```

```
}
```

## 2.2.4 Status

The method returns the Official Confirmation of Receipt of uploaded documents. This method is a part of API for the clients, available at the level of the same service as the other methods.

| Name                  | Status                                  |
|-----------------------|---|
| Method type           | GET                                     |
| Uploaded content type | Query String                            |
| Returned content type | application/json                        |
| Maximum request size  | 100KB                                   |
| Format                | Status/ba96951d00635700000001726b6ec621 |

**Description of uploaded parameter:**

| Name            | Description                           | Type   | Validation |
|-----------------|---------------------------------------|--------|------------|
| ReferenceNumber | Reference Number – Session identifier | String | Required   |

**The Status method returns three types of responses:**

| Response code      | Description                               |
|--------------------|---|
| 200 – OK           | Confirmation returned successfully        |
| 400 – Bad Request  | Erroneous request. Erroneous service call |
| 500 – Server Error | Erroneous request processing              |

**Description of JSON (application/json) structure of valid response (200 – OK):**

| Name               | Description                                  | Type     |
|--------------------|--|----------|
| <b>Code</b>        | Status code                                  | String   |
| <b>Description</b> | Description                                  | String   |
| <b>Details</b>     | Event details                                | String   |
| <b>Upo</b>         | Optionally. Official confirmation of receipt | String   |
| <b>Timestamp</b>   | Time stamp                                   | Datetime |

Example:

```
{
  "Code": 300,
  "Description": "Invalid reference number",
  "Upo": "",
  "Details": "",
  "Timestamp": "2016-06-17T09:37:40.773976+00:00"
}
```

**List of statuses:**

The table below presents the status codes and their descriptions returned in a valid response using the Status method. The statuses are grouped as follows:

1xx – Codes specifying the session status situations (e.g. initiated, expired)

2xx – Codes specifying successful completion of document processing

3xx – Codes informing on the document processing stage

4xx – Codes specifying failed completion of document processing

| Status code | Description   |
|-------------|---|
| <b>100</b>  | File uploading session initiated  |
| <b>101</b>  | X of Y declared files received  |
| <b>120</b>  | Session completed successfully. Data saved correctly. Document verification in progress |
| <b>200</b>  | Document processing completed successfully, download UPO                                |
| <b>300</b>  | Invalid reference number  |

| Status code | Description   |
|-------------|---|
| 401         | Verification negative – document incompatible with the XSD scheme   |
| 405         | Document with revoked certificate   |
| 406         | Document with certificate with non-supported provider   |
| 407         | Document duplicate uploaded. Reference number of the original document is XXXXXXXX  |
| 408         | Document contains errors preventing its processing  |
| 410         | Uploaded files are not valid ZIP archive  |
| 411         | Verification negative – identical document is already uploaded in the system  |
| 412         | Document encryption invalid   |
| 413         | Document checksum incompatible with the declared value  |
| 415         | Uploaded document type is not supported by the system   |
| 417         | Document encryption invalid. Authorization data decryption error  |
| 418         | Verification negative – authorization data incompatible with the XSD scheme   |
| 419         | Verification negative – authorization data error  |
| 420         | No valid power of attorney/authorisation to sign document   |
| 422         | Verification negative – document uploaded with the use of authorization data may be sent only by a tax payer being a natural person |
| 423         | Document with a certificate having no required attributes   |
| 424         | Verification negative – document cannot be signed with the use of authorization data  |
| 425         | Verification negative – inconsistent data   |
| 426         | Invalid character encoding in authorization data  |
| 427         | Document with certificate with invalid URL  |
| 428         | Business rules validation error   |
| 430         | Document with invalid signature   |

#### Description of JSON (application/json) response (400 – Bad Request):

| Name             | Description                   | Type        |
|------------------|-------------------------------|-------------|
| <b>Message</b>   | Error message                 | String      |
| <b>Errors</b>    | Optionally. Error table       | String list |
| <b>RequestId</b> | Unique bad request identifier | GUID        |

Example:

```
{
  "Message": "Bad request",
  "RequestId": "172dc3cc-5b97-48de-91dd-6903587cba19"
}
```

#### Description of JSON (application/json) response (500 – Internal Server Error):

| Name             | Description                   | Type   |
|------------------|-------------------------------|--------|
| <b>Message</b>   | Error message                 | String |
| <b>RequestId</b> | Unique bad request identifier | GUID   |

Example:

```
{
  "Message": "Internal system error",
  "RequestId": "172dc3cc-5b97-48de-91dd-6903587cba19"
}
```