Podpisuj

Signature API

www.archimetes.com

2019-09-25

Table of Contents

About

Signature API allows document signing using desktop application `Podpisuj` and offers a possibility for integration partners to integrate electronic signing of documents into their business processes.

The application listens on port `14741` for regular HTTP `GET` or `POST` requests. The response is always an `application/json` formatted HTTP response.

All requests, except status and logout, require:

- a user with the `API` function set,
- an authorization code (see annex Examples for details).

The remaining (method specific) mandatory parameters are denoted with bold in the text below.

Application status

Method `GET http://127.0.0.1:14741` allows the caller to retrieve basic application information as well as information about the current user and the user's certificate (if selected).

Input: None.

Output: Application and user/tenant information (if logged in).

```
{
    "appInfo": {
        "name": "Signer",
        "version": "3.300",
        "isInUse": false,
        "isInApiMode": false
    },
    "accountInfo": {
        "user": null,
        "tenant": null
    },
    "otherInfo": {
        "userCertificate": null
    }
}
```

```
{
    "appInfo": {
        "name": "Signer",
        "version": "3.300",
        "isInUse": true,
        "isInApiMode": false
    },
    "accountInfo": {
        "user": {
            "email": "user1@acme.org",
            "firstName": "firstname",
            "lastName": "lastname",
            "isApiUser": true
        },
        "tenant": {
            "name": "acme"
        }
    },
    "otherInfo": {
        "userCertificate": null
    }
}
```

Login

Method `POST 127.0.0.1:14741/login` allows to login a (partner’s) user.

Input:

- **username**
- **partnerId**
- **authorizationCode**
- orgId

Example:

```
curl -X POST
'http://127.0.0.1:14741/login?username=myusername&authorizationCode=mycode&partnerId=m
ypartnerid'
```

Output: None (use application status method to retrieve user data).

Logout

Method `POST` `127.0.0.1:14741/logout` signs out the currently logged in user (if any).

Input: None.

Output: None.

Selecting user certificate

Method `POST` `http://127.0.0.1:14741/selectcertificate` allows to set the user's certificate.

Input:

- **authorizationCode**

Output: User certificate information (if set) or empty json (if the user cancelled the selection, note that this has no effect on the currently set certificate, retrievable by the GET method).

Document signing

Method `POST` `http://127.0.0.1:14741/sign` facilitates the signing of one or more documents.

Input:

- **documents to sign** (sent as `multipart/form-data`)
- **authorizationCode**

Example:

```
curl -X POST http://127.0.0.1:14741/sign?authorizationCode=mycode \
  -H 'content-type: multipart/form-data; boundary=----
WebKitFormBoundary7MA4YWxkTrZu0gW' \
  -F 'file1=@C:\Temp\HelloWorld1.txt' \
  -F 'file2=@C:\Temp\HelloWorld2.txt'
```

Output: Array containing signed documents:

```
{
   "files":[
      {
         "name":"<filename1>",
         "mediaType":"<mediaType>",
         "size":"<size in bytes>",
         "metadata":{"pyPdfIsEncrypted":"<true or false>","pyNumberOfPages":"<number
of pages>"},
         "binaryData":"<base64 data>"
      },
      {
         "name":"<filename2>",
         "mediaType":"<mediaType>",
         "size":"<size in bytes>",
         "metadata":{"pyPdfIsEncrypted":"<true or false>","pyNumberOfPages":"<number
of pages>"},
         "binaryData":"<base64 data>"
      }
   ]
}
```

An empty JSON may be returned to indicate that the user cancelled the activity and did not sign some or all documents.

Annex - Examples

Some requests, like getting basic application info or logout, do not need any parameters and may be called as is e.g. by calling the url http://127.0.0.1:14741 from your browser.

Some requests require an authorization code to execute to ensure that the user is authenticated and to validate the origin of the request. The authorization code has following format:

`<unix timestamp>:<signed unix timestap>`

where:

- unix timestamp is the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC (e.g. `System.currentTimeMillis()` in Java or `Date.now()` in Javascript).
- signed unix timestamp is RSA-SHA256 (base64 encoded) signature of the timestamp performed by the private key of the user (or the "to be" user in case of login method)

Example:

```
1565381254347:NVGqZe......4Zig==
```

The code can be passed to the application by following means:

- as a parameter `authorizationCode` in query string (in this case the whole authorization code must be URL encoded),
- as part of other form-data with name `authorizationCode`,
- as `X-Authorization-Code` header (this method is now deprecated and supported for backward compatibility only and should not be used because of CORS related issues in browsers).

To automate the generation of the authorization code see examples in the following sub chapters.

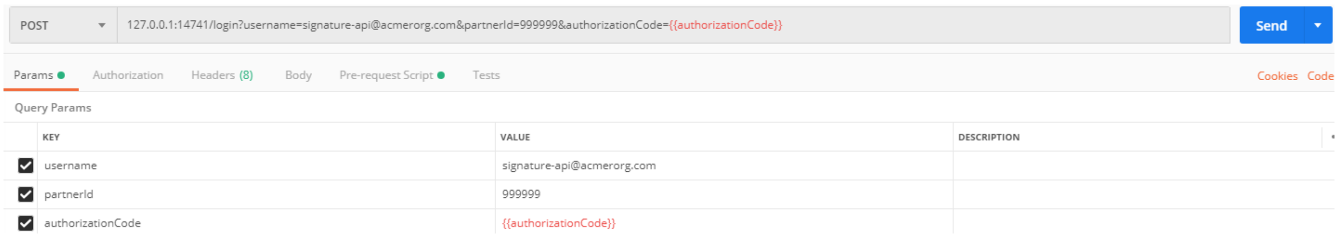
Postman

For Postman generated requests, use the code for the pre-request script below. As the crypto-js does not support RSA we'll use the https://github.com/archimetes/RSAForPostman forge library adaptation. The library file must be hosted on a web server or put as a single variable into the script. The private key belongs to the user as generated by the partner API. **The private key must be kept secret on the server side** and **the signining must be performed on the server side** as well. The value used in the code below must be replaced with the real value before testing.

```
// Download forgeJS from the web and set it as a variable
if(!pm.globals.has("forgeJS")){

pm.sendRequest("https://raw.githubusercontent.com/archimetes/RSAForPostman/master/forg
e.js", function (err, res) {
        if (err) {
            console.log(err);}
        else {
            pm.globals.set("forgeJS", res.text());}
})}
eval(postman.getGlobalVariable("forgeJS"));
// Note: if you do not want to use a web server, you can simply create a global
variable named forgeJS and copy content of this file into this script.

// Set private key
var privateKeyPEM = "-----BEGIN PRIVATE KEY-----\
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCRjcMgxXzpT6g/\
...\
pl+uS3h7lwI5Bd2J3kKGOFIP\
-----END PRIVATE KEY-----";
var privateKey = forge.pki.privateKeyFromPem(privateKeyPEM);

// Set public key (openssl rsa -in privateKey.pem -pubout > publicKey.pub)
var publicKeyPEM = "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkY3DIMV86U+oP9b6e+Ax\
...\
tQIDAQAB\
-----END PUBLIC KEY-----";
var publicKey = forge.pki.publicKeyFromPem(publicKeyPEM);

// Get a timestamp
var currentTime = Date.now() + "";

// Sign data
var md = forge.md.sha256.create();
md.update(currentTime);

var signature = privateKey.sign(md);
console.log("Signature:" + forge.util.encode64(signature));

// Verify signature (pro-forma)
var verificationResult = publicKey.verify(md.digest().getBytes(), signature);
console.log("Verification:" + verificationResult);

// Create authorization code
var authorizationCode = currentTime + ":" + forge.util.encode64(signature);
console.log("Authorization code:" + authorizationCode);
authorizationCode = encodeURIComponent(authorizationCode);
console.log("Authorization cod (URI encoded):" + authorizationCode);

pm.variables.set("authorizationCode", authorizationCode);
```

The authorization code is set as a variable that can be used in the Postman requests.

Login method:

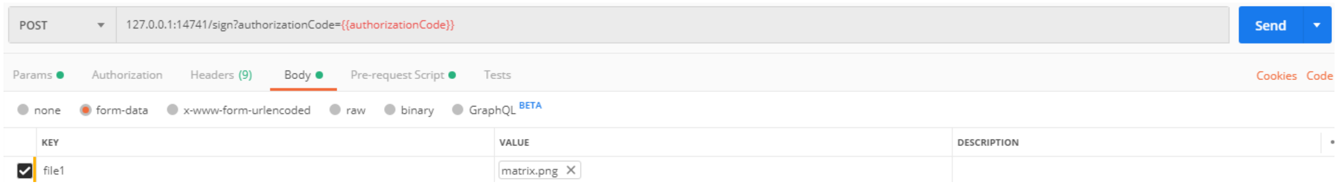

POST 127.0.0.1:14741/login?username=signature-api@acmerorg.com&partnerId=999999&authorizationCode={{authorizationCode}}

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests

Query Params

KEY	VALUE	DESCRIPTION
✓ username	signature-api@acmerorg.com	
✓ partnerId	999999	
✓ authorizationCode	{{authorizationCode}}	

Sign method (the authorization code is set as in the previous example, but putting it in the form-data is also possible):


POST 127.0.0.1:14741/sign?authorizationCode={{authorizationCode}}

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests

none | ● form-data | x-www-form-urlencoded | raw | binary | GraphQL BETA

KEY	VALUE	DESCRIPTION
✓ file1	matrix.png	

Java

For Java use standard JDK functions (javax.crypto.*) for cryptography.

```
String msg =  System.currentTimeMillis()+"";
String keyString = "MIIEvA....x04g==";
PKCS8EncodedKeySpec spec = new
PKCS8EncodedKeySpec(Base64.getMimeDecoder().decode(keyString));
KeyFactory kf = KeyFactory.getInstance("RSA");
PrivateKey privateKey = kf.generatePrivate(spec);
Signature signature = Signature.getInstance("SHA256withRSA");
signature.initSign( privateKey );
signature.update(msg.getBytes());
byte[] sign = signature.sign();
System.out.println(msg+":"+ Base64.getEncoder().encodeToString( sign));
```

Annex - Error list

If error occurs and the request can’t be served or return value can’t be obtained one of the following HTTP codes is returned:

- `UNAUTHORIZED` - authenticated user is necessary to process this request. Use the `login` call to login,
- `FORBIDDEN` - authenticated user does not have necessary permissions to make this call (probably the feature is not available for user’s organization),
- `UNSUPPORTED_MEDIA_TYPE` - input data/file has an unsupported media type,
- `SERVICE_UNAVAILABLE` - application is busy serving another request,
- `BAD_REQUEST` - input data is incomplete or ill-formatted.,
- `INTERNAL_SERVER_ERROR` - internal application error,

For some errors, there may be additional JSON formatted data regarding the cause of the error in the following structure:

```
{
  "error": {
    "cause": "<Exception type>",
    "message": "<Exception message or description>"
  }
}
```