THE Berlin GROUP

A EUROPEAN STANDARDS INITIATIVE

DRAFT

openFinance API Framework

Implementation Guidelines
HTTP Message with Signed Body

(to be integrated into [oFA-ProtSec] after consultation)

Last update: 12.07.2023

License Notice

This Specification has been prepared by the Participants of the openFinance Taskfoce*. This Specification is published by the Berlin Group under the following license conditions:

- "Creative Commons Attribution-NoDerivatives 4.0 International Public License"

  

  This means that the Specification can be copied and redistributed in any medium or format for any purpose, even commercially, and when shared, that appropriate credit must be given, a link to the license must be provided, and indicated if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. In addition, if you remix, transform, or build upon the Specification, you may not distribute the modified Specification.

- Implementation of certain elements of this Specification may require licenses under third party intellectual property rights, including without limitation, patent rights. The Berlin Group or any contributor to the Specification is not, and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.
- Any right, title and interest in and to the copyright and all related rights in topic-related Scheme Rulebooks, belong to the respective Scheme Manager (amongst others, the European Payments Council AISBL - EPC).
- The Specification, including technical data, may be subject to export or import regulations in different countries. Any user of the Specification agrees to comply strictly with all such regulations and acknowledges that it has the responsibility to obtain licenses to export, re-export, or import (parts of) the Specification.

* The openFinance Taskforce brings together participants of the Berlin Group with additional European banks (ASPSPs), banking associations, payment associations, payment schemes and interbank processors.

Contents

1 Introduction

For V2 of the openFinance API Framework HTTP (request or response) messages with signed bodies will be introduced.

For an HTTP message with a signed body, only the content of the body shall be signed. One or more signatures to may be applied to the content of the body may be possible.

HTTP messages with a signed body should not be confused with signed HTTP messages (as defined by section 6 of [oFA-ProtSec]). For an HTTP request message with a signed body a signature is generated by a PSU. In this case the signature by the PSU is considered as a possible method of an SCA by the PSU to authorise the transaction. For signed HTTP messages the HTTP message is signed by the TPP as a proof of origin of the message.

For messages sent by the ASPSP, a signature for the content of the body could be needed from a contractual point of view end to end, e.g. sending signed account statements or in future contract proposals.

This document is an excerpt from the document [oFA-ProtSec], which has already been part of the consultation process. After consultation of the content of section 2 of this document at hand it will be integrated into [oFA-ProtSec] as section 7.1.

2 Signing the body of an HTTP message

Remark: To be integrated into [oFA-ProtSec] as section 7.1.

The body of a request or response message may be signed by one or more responsible entities. Who is responsible for signing the body depends on the single use case. The following are possible examples:

- For a request message sent by an API Client to the openFinance API of an ASPSP the body may be signed by one or more PSU.
- For a response message to a request of an API Client the body may be signed by a responsible department or one or more responsible employees of the ASPSP.
- For a request message as part of a pushed-based service the body may be signed by a responsible department or one or more responsible employees of the ASPSP.

If the body of a request message sent by an API Client to the openFinance API of an ASPSP is signed by one or more PSU this will be called a **signed payment request**, if the request message is part of a payment service and contains payment data. For a signed payment request further SCA procedures of the PSU involved will not be necessary if the creation of the corresponding signatures is compliant with the requirements of the RTS.

Remarks:

- Signing the body of a request message for a signed payment request should not be confused with the signing of an HTTP request message. Signing the HTTP message will authenticate the API Client sending the request message to the openFinance API of the ASPSP. It will not authenticate the PSU and for this reason cannot substitute an SCA of the PSU.
- To distinguish the case of a request message with a signed body (indicating the consent of the PSU) from signed HTTP messages (indicating the proof of origin by the TPP) these request messages with signed bodies will be called in general **signed transaction requests**.

It is up to the ASPSP to decide if it supports signed bodies for messages. For some use cases an ASPSP might mandate that the body of a request message sent by an API Client has to be signed by one or more PSUs.

This specification does not define any requirements about the quality of the process to generate the signature over the body of the http message. It is possible that the ASPSP will define further requirements for this as for example

- special algorithms to be used,
- key length to be used,
- quality of certificates to be used.

Different signature procedures and profiles based on asymmetric and symmetric cryptography will be supported in future by this specification. The phrase "signing a body" will be used regardless if the body will be secured by an electronic signature or an electronic seal based on asymmetric cryptography or by some kind of cryptogram based on symmetric cryptography, as for example an EMV cryptogram generated by a payment card.

2.1 Extension to the http message

Remark: To be integrated into [oFA-ProtSec] as section 7.1.1

Remark: The definitions of this section 2.1 hold regardless which signature profile is used to sign the body according to section 2.2 (JAdES_JS), 2.3 (XAdES) or 2.4 (EMV_AC).

2.1.1 Path

For sending request messages with a signed body dedicated endpoints will be used.

Example: For a signed payment request the endpoint

POST …/v2/signed-payments/{payment-product}

has to be used instead of

POST …/v2/payments/{payment-product}

For other (extended) payment services or other services the endpoints have to be adapted accordingly with an "signed-" prefix if the request is sent with a signed body.

These dedicated endpoints for request messages with signed bodies shall not be used for request message if the body is not signed, since the schemas for the definition of the (data structures of the) bodies and also the following steps for the authorisation will be different.

2.1.2 Query parameters

None.

2.1.3 Header parameters

If the body is signed the following header parameters shall be added to the header of the HTTP request or response message:

Attribute	Type	Condition	Description
Body-Sig-Profile	String	Conditional	Indicates the signature profile used for signing (parts of) the body. Shall be used if the body is signed.

Table 1: Header parameters to indicate that the body has been signed.

Body-Sig-Profile

This parameter indicates the profile used for signing the message body.

This header parameter is mandatory if the body of the message is signed. It shall not be used if the body of the message is not signed.

Currently the following values are supported by this specification:

JAdES_JS	The body is signed based on [RFC7515] using JWS JSON Serialization taking the requirements of [ETSI TS 119 182-1] for JAdES into account.
XAdES	The body is signed based on [W3C XMLSig] taking the requirements of [ETSI EN 319 132-1] for XAdES into account.
EMV_AC	FOR FUTURE USE ONLY (The body is signed using an EMV AC cryptogram).

Table 2: Supported signature profiles.

Future versions of the specification may support further values for the signature profile.

Remark:

For signed transaction requests not all header parameters as defined by section 3.4 of [oFA-ProtSec] may be applicable.

2.1.4 Body

For an unsigned JSON coded body, the content of the unsigned body has to be replaced by a signed data structure as described in section 2.2.

For an unsigned XML coded body, the content of the unsigned body has to be replaced by a signed data structure as described in section 2.3.

Other cases are RFU.

2.2 Signing the body using JAdES_JS

Remark: To be integrated into [oFA-ProtSec] as section 7.1.2

The profile JAdES_JS can only be used to sign the body of a message if the content of the body is JSON encoded.

Only general JWS JSON Serialisation according to section 7.2.1 of [RFC7515] is supported. By this the body of the message may be protected by one or more electronic signatures (which would not be available for compact serialisation).

For each signature to protect the content of the body a JWS has to be generated using the key and the certificate of the responsible PSU.

Unsigned body:

```
{ JSON encoded unsigned body data }
```

Remark: The body may contain more than one JSON coded elements on the first level.

Signed body:

A new data element to "envelop" the unsigned body is introduced. This element is called `payload`. This new element will be a sibling element to the signatures in the signed body.

```
{
      "payload": { JSON encoded unsigned body data },
      "signatures": [ array of JWS ]
}
```

Table 3: Elements of a signed JSON message body for alternative B.

Remarks:

- If the message is sent to an endpoint dedicated for signed request messages the elements `payload` and `signatures` are mandatory elements and are the only JSON coded elements on the first level.
- If the message is sent to another endpoint the elements `payload` and `signatures` shall not be used.
- The array element *signatures* shall contain one JWS for each signature generated to protect the body.

Only detached signatures signing local data are used. The data to be signed can be protected by n signatures (with n >= 1). Note that these are independent signatures and not counter signatures. If one signature has to be protected by another signatures, counter signatures as described in section 5.3.2 of [ETSI TS 119 182-1] shall be used.

If the body is protected only by a single signature, the flattened syntax according to section 7.2.2 of [RFC7515] may be used.

Question for the consultation process: Should the flattened syntax be supported, eg. for the access via wallets? If yes this has to be considered by the definition of the yaml-files.

Detached signatures are built according to appendix F of [RFC7515], i.e. the JWS does not contain the `payload` element. Instead, the signature protects the content of the element `payload` being the only sibling element of the element `signatures` within the signed body of the HTTP message.

For the creation and verification of a signature the following element has to be considered (as virtual element of the JWS but not added to the JWS):

```
"payload":"BASE64URL(JWS Payload)"
```

with JWS Payload defined by JSON encoded content of the sibling element `payload` of signatures within the body of the HTTP message.

Each JWS is the following JSON structure:

Element	Type	Condition	Description
protected	String	Mandatory	Base64URL encoded JWS Protected Header. BASE64URL(JWS Protected Header)
header	String	Optional	Content of the JWS Unprotected Header.
signature	String	Mandatory	Base64URL encoded JWS Signature. BASE64URL(JWS Signature)

Table 4: JSON structure of a JWS

JWS Protected Header

The JWS Protected Header is mandatory, i.e. the element protected shall be part of the JWS. It shall contain at least the following sub elements:

Element	Type	Condition	Description
alg	String	Mandatory	Identifier of the algorithm used for the creation of the signature according to [RFC7518]. Examples: RS256, PS256

Element	Type	Condition	Description
x5c	String	{Or	The "x5c" (X.509 certificate chain) Header Parameter contains the X.509 public key certificate or certificate chain corresponding to the key used to generate the signature. The certificate or certificate chain is represented as a JSON array of certificate value strings. Each string in the array is a base64-encoded (**not** base64url-encoded) DER PKIX certificate value. The certificate containing the public key corresponding to the key used to generate the signature MUST be the first certificate. This MAY be followed by additional certificates, with each subsequent certificate being the one used to certify the previous one. The recipient MUST validate the certificate chain according to RFC 5280 and consider the certificate or certificate chain to be invalid if any validation failure occurs.
x5u	String	Or}	The "x5u" (X.509 URL) Header Parameter is a URI that refers to a resource for the X.509 public key certificate or certificate chain corresponding to the key used to generate the signature. The identified resource MUST provide a representation of the certificate or certificate chain that conforms to RFC 5280 in PEM-encoded form, with each certificate delimited as specified in Section 6.1 of RFC 4945. The certificate containing the public key used to generate the signature MUST be the first certificate. This MAY be followed by additional certificates, with each subsequent certificate being the one used to certify the previous one. The protocol used to acquire the resource MUST provide integrity protection; an HTTP GET request to retrieve the certificate MUST use TLS; and the identity of the server MUST be validated, as per Section 6 of RFC 6125.

Table 5: JSON structure of a JWS Protected Header

Remarks:

- The ASPSP can mandate algorithms to be supported for the creation of a signature.
- One of the elements x5u or x5c shall be contained in the JWS Protected Header.
- The ASPSP may define further restriction for the content of the elements x5u and x5c. For example, the ASPSP may require that the element x5c is contained and

that this element contains only the certificate itself but not the complete certificate chain.

- Further elements defined in section 5.1 and 5.2 of [ETSI TS 119 182-1] may be included. The ASPSP may define additional requirement on these elements, i.e. that an element shall be included or that an element shall not be included.

- All additional requirements defined by the ASPSP have to be described by the ASPSP specific documentation.

JWS Unprotected Header

The JWS Unprotected Header is optional, i.e. the element header may be missing. If the element header exists in the JWS, it shall contain only one sub element etsiU according to section 5.3.1 of [ETSI TS 119 182-1].

If a signature A has to be secured by another signature B, i.e. a counter signature according to [ETSI TS 119 182-1], the JWS representing the counter signature B shall be enclosed in the unprotected header of the JWS representing the signature A. In this case the sub element etsiU of the element header (containing the JWS unprotected header) shall contain the element sigC containing the JWS representing the counter signature.

JWS Signature

The signature is calculated over the string

ASCII(BASE64URL(UTF8(JWS Protected Header)) || '.' || BASE64URL(JWS Payload))

using the private key corresponding to the certificate contained in the element x5c or referenced by the URI contained in x5u using the algorithm identified by the element alg.

2.3 Signing the body using XAdES

Remark: This section is already contained in [oFA-ProtSec] as section 7.1.3 without any changes.

The profile XAdES can be used to sign the body of a message regardless of the coding of the content. Nevertheless, in the following only the case of an XML coded body is considered.

ETSI European Norm [ETSI EN 319 132-1] and [W3C XMLSig] will be used for the XAdES profile. Note that for the XML Signature Syntax and Processing also a newer version exists [W3C XMLSig V2], but for compliance reasons with the European Norm version 1.1 as defined by [W3C XMLSig] is used.

The body of the message will be replaced by an XML document representing the signed body as follows:

Unsigned body:

```
<any_tag> body data to be signed </any_tag>
```

Signed body:

```
<SignedBody>
      <Object Id="ID_bodyToBeSigned">
            <any_tag> body data to be signed </any_tag>
      </Object>
      <Object>
            <Manifest Id="ID_manifest"> … </Manifest>
      </Object>
      </Signature Id="ID_signature_1"> … </Signature>
      …
      </Signature Id="ID_signature_n"> … </Signature>
</SignedBody>
```

Table 6: Elements of a signed XML message body.

Namespaces have to be included as described by [ETSI EN 319 132-1] and [W3C XMLSig].

Only detached signatures signing local data are used, i.e. the data to be signed is contained in a sibling element. The data to be signed can be protected by n signatures (with n >= 1). Note that these are independent signatures and not counter signatures. If one signature has to be protected by another signatures, counter signatures as described in section 5.2.7 of [ETSI EN 319 132-1] shall be used.

The manifest is introduced to increase efficiency for the generation and verification of the signatures. The hash value over the body data to be signed has to be calculated only once and not separately for each signature.

The IDs shown above are only place holders. During creation of the XML document it has to be taken care that the concrete values used for these IDs do not produce any collisions that violate the ID uniqueness.

Algorithms

For transformation, canonicalization, hash value calculation and signature creation the algorithms have to be supported as defined by [ETSI EN 319 132-1]. Out of this set of algorithms the ASPSP can mandate algorithm to be used.

Element Manifest

The element Manifest has got the following content:

```
<Manifest Id="ID_manifest>
     <Reference URI="#ID_bodyToBeSigned">
          <Transforms>
               <Transform Algorithm="URI_TransformAlgorithm"/>
          </Transforms>
          <DigestMethod Algorithm="URI_HashAlgorithm"/>
          <DigestValue> base64 coded hash value </DigestValue>
     </Reference
</Manifest>
```

Table 7: Content of the element Manifest.

This manifest contains only the digest value for the body to be signed. It will be referenced in each element containing a signature. This manifest is only created once regardless how many signatures will be generated to protect the body. Also, during verification of the signatures, the content of this manifest is only created once.

Element Signature

An element Signature containing a single signature protecting the body has got the following content:

```
<Signature Id="ID_signature_k">
     <SignedInfo> … </SignedInfo>
     <SignatureValue>
          base64 coded signature value
     </SignatureValue>
     <KeyInfo> … </KeyInfo>
     <Object>
          <QualifyingProperties>
               <SignedProperties Id="ID_signedProp_k"> …
               </SignedProperties>
```

```
                    <UnsignedProperties> … </UnsignedProperties>
                </QualifiedProperties>
            </Object>
        </Signature>
```

Table 8: Content of a single signature element

Element QualifyingProperties

The Element QualifyingProperties contains the signed and not signed properties of the data to be signed and of the signature according to [ETSI EN 319 132-1]. The element SignedProperties will be referenced in the element SignedInfo and by this protected by the signature contained in the element SignatureValue.

The signature has to be compliant at least with XAdES baseline signatures of level B-B defined in section 6 of [ETSI EN 319 132-1]. For this reason, the element SignedProperties may not be empty (see section 6.3 of [ETSI EN 319 132-1]). It has got the following sub elements:

```
<SignedProperties Id="ID_signedProp_k">
    <SignedSignatureProperties> …
    </SignedSignatureProperties>
    <SignedDataObjectProperties> …
    </SignedDataObjectProperties>
</SignedProperties>
```

Table 9: Content of the element SignedProperties

The element SignedSignatureProperties has to contain at least the following sub elements:

- SigningTime (see section 5.2.1 of [ETSI EN 319 132-1]).
- SigningCertificateV2 (see section 5.2.2 of [ETSI EN 319 132-1]).

The element SignedDataObjectProperties hat to contain at least the following sub elements:

- DataObjectFormat (see section 5.2.4 of [ETSI EN 319 132-1]) containing at least the MIME type indicating the format of the body data to be signed.

The element UnsignedProperties may be empty. If it is empty it is missing. Empty elements are not allowed according to [ETSI EN 319 132-1].

An ASPSP can mandate higher levels defined by section 6 of [ETSI EN 319 132-1] depending on the nature of the service.

Element SignedInfo

The element SignedInfo contains the information which data is signed (data of the unsigned body and the signed properties) and how the signature has to be created. It has got the following content:

```
<SignedInfo>
     <CanonicalizationMethod Algorithm= URI_CanoniAlgorithm/>
     <SignatureMethod Algorithm="URI_SignatureAlgorithm"/>
     <Reference URI="#ID_bodyToBeSigned">
          Type="http://www.w3.org/2000/09/xmldsig#Manifest">
          <Transforms>
               <Transform Algorithm="URI_TransformAlgorithm"/>
          </Transforms>
          <DigestMethod Algorithm="URI_HashAlgorithm"/>
          <DigestValue> base64 coded hash value </DigestValue>
     </Reference>
     <Reference URI="#ID_signedProp_k"
          <Transforms>
              <Transform Algorithm="URI_TransformAlgorithm"/>
          </Transforms>
          <DigestMethod Algorithm="URI_HashAlgorithm"/>
          <DigestValue> base64 coded hash value </DigestValue>
     </Reference>
</SignedInfo>
```

Table 10: Content of the element SignedInfo

Element KeyInfo

The element KeyInfo contains information about the key needed for the verification of the signature. According to section 6.3 of [ETSI EN 319 132-1]

- it **shall** contain at least the signing certificate, i.e. the X.509 certificate belonging to the public key needed for the verification of the signature, and
- it **should** contain all certificates not already available to the relying party needed to verify the signing certificate.

The element has got at least the following content:

```
<KeyInfo>
     <X509Data>
          <X509Certificate>
               base64 coded X.509 signer certificate
          </X509Certificate>
     </X509Data>
</KeyInfo>
```

Table 11: Content of the element KeyInfo

The sub element X509Data may contain more than one certificate building a path from the signer certificate to a root certificate or to a CA contained in a trusted list.

2.4 Signing the body using EMV_AC

Remark: This section is already contained in [oFA-ProtSec] as section 7.1.4 without any changes.

Not supported for the current version of the openFinance Framework. Support will be added as part of future versions.

3 References

3.1 Documents of the NextGenPSD2 XS2A Framework

[XS2A-SecB] NextGenPSD2 XS2A Framework, Security Bulletin, Version 1.1, 30 October 2020

3.2 Documents of the openFinance API Framework

[oFA-OR-ADM] openFinance API Framework, Operational Rules for Administrative Service, Version 0.9, published for market consultation 26 May 2021

[oFA-PDM-V2] openFinance API Framework, Payment Data Model for Version 2.x, version 1.0, 24 September 2021

[oFA DD] openFinance API Framework, Data Dictionary for V2.x, Draft 5 April 2023

[oFA-ProtSec] openFinance API Framework, Implementation Guidelines, Protocol Functions and Security Measures, in preparation

3.3 Further documents

[eIDAS] Regulation (EU) No 910/2014 of the European Parliament and of the Council on Electronic Identification and Trust Services for Electronic Transactions in the Internal Market, 23 July 2014, published 28 August 2014

[ETSI EN 319 132-1] ETSI European Standard, Electronic Signatures and Infrastructures (ESI); XAdES digital signatures; Part 1: Building blocks and XAdES baseline signatures, V1.1.1 (2016-04)

[ETSI TS 119 182-1] ETSI Technical Specification, Electronic Signatures and Infrastructures (ESI); JAdES digital signatures; Part 1: Building blocks and JAdES baseline signatures, V1.2.1 (2022-02)

[ETSI TS 119 495] ETSI Technical Specification, Electronic Signatures and Infrastructures (ESI); Sector Specific Requirements; Certificate Profiles and TSP Policy Requirements for Open Banking, V1.5.1 (2021-04)

[RFC3230] Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, DOI 10.17487/RFC3230, January 2002,https://www.rfc-editor.org/info/rfc3230

[RFC7515] Jones, Bradley, Sakimura, "JSON Web Signatures (JWS)", May 2015, https://datatracker.ietf.org/doc/rfc7515/

[RFC7518] Jones, "JSON Web Algorithms (JWA)", May 2015, https://datatracker.ietf.org/doc/rfc7518/

[RFC 7797] Jones, "JSON Web Signature (JWS) Unencoded Payload Option", February 2016, https://datatracker.ietf.org/doc/rfc7797/

[W3C XMLSig] W3C Recommendation: "XML Signature and Processing", Version 1.1, 11 April 2013

[W3C XMLSig V2] W3C Recommendation: "XML Signature and Processing", Version 2.0, 23 July 2015