Architectures and protocols for remote signature applications

Contents

Foreword

This document is a work by members of the Cloud Signature Consortium, a nonprofit association founded by industry and academic organizations for building upon existing knowledge of solutions, architectures and protocols for Cloud-based Digital Signatures, also defined as “remote” Electronic Signatures.

The Cloud Signature Consortium has developed the present specification to make these solutions interoperable and suitable for uniform adoption in the global market, in particular – but not exclusively – to meet the requirements of the European Union's Regulation 910/2014 on Electronic Identification and Trust Services (eIDAS) [i.1], which formally took effect on 1 July 2016.

Revision history

Version	Date	Version change details
0.1.7.9-PR	14/02/2017	Public Pre-Release for early implementations
1.0.2.4-PR	24/09/2018	V1 Pre-Release for public comments
1.0.3.0	13/12/2018	V1 Public Release
1.0.4.0	28/06/2019	V1 Updated with new IPR information and errata
2.0.0.0	25/03/2022	V2 Pre-Release for public comments
2.0.0.1	19/08/2022	V2 Pre-Release after solving public comments

Acknowledgements

This work is the result of the contributions of several individuals from the Technical Working Group of the Cloud Signature Consortium and some additional contributors. In particular, the following people have provided a significant contribution to the drawing up and revision of the present specification:

Ałła Stoliarowa-Myć, Andrea Röck, Andrea Valle, Andrew Papastefanou, Andreas Vollmert, Arno Fiedler, Bernd Wild, Carlos Ares, Cornelia Enke, Daniel Fett, David Ruana, Davide Barelli, Enrico Entschew, Francesco Barcellini, Franck Leroy, Giuliana Marzola, Giuseppe Damiano, Harald Bratko, Håvard Grindheim, Iñigo Barreira, Jon Ølnes, Kapil Khattar, Dr. Kim Nguyen, Klaus-Dieter Wirth, Luca Boldrin, Luigi Rizzo, Mangesh Bhandarkar, Marc Kaufman, Marcin Szulga, Meena Muralidharan, Michael Traut, Patrycja Wiktorczyk, Patryk Sosiński, Peter Lipp, Prof. Reinhard Posch, Thomas Pielczyk, Torsten Lodderstedt.

Introduction

For a long time, transactional e-services have been designed for typical end-user devices such as desktop computers and laptops. Accordingly, existing digital signature solutions are tailored to the characteristics of these devices as well. This applies to smart card and USB token-based solutions. These traditional signature solutions implicitly assume that the user accesses e-services from a desktop or laptop computer and in addition uses a smart card or token to create any required digital signatures. This assumption is not valid any longer. During the past few years, smartphones, tablets and other mobile end-user devices have started to replace desktop and laptops computers.

This situation raises several challenges for e-services: smart cards and tokens cannot be easily connected to smartphones and other mobile devices, or cannot at all. For instance, smartphones usually do not provide support for USB devices, which is the common technology for smart card based solutions.

In this regard, recent regulations in various regions worldwide – like eIDAS [i.1] in the European Union – have introduced the concept of electronic signatures that are created using a “remote signature creation device”, which means that the signature device is not anymore a personal device under the physical control of the user, but rather it is replaced by cloud-based services offered and managed by a trusted service provider.

This is, in summary, the scope of the Cloud Signature Consortium, also known as CSC, aiming at the definition of a common architecture, building blocks and communication protocols intended for creating a standard API to integrate the essential components of a remote signature solution established among different service providers and consumers.

Where the context of the eIDAS Regulation is applicable, this specification, and the term “remote signature solution” herein developed, aim to cover solutions for remote electronic signatures and remote electronic seals, in the domains of both qualified and advanced electronic signatures / seals.

Intellectual Property Rights

The Intellectual Property Rights Policy (IPR Policy) of the Cloud Signature Consortium is available at https://cloudsignatureconsortium.org/ipr/.

Trademark notice

The Cloud Signature Consortium logo is a Registered Trademark of the Cloud Signature Consortium: EU Trademark number 015579048.

Essential Patents

IPRs essential or potentially essential to the present document may have been declared to the Cloud Signature Consortium. The information pertaining to these essential IPRs, if any, is available on request from the Cloud Signature Consortium secretariat at info@cloudsignatureconsortium.com.

No investigation, including IPR searches, has been carried out by the Cloud Signature Consortium. No guarantee can be given as to the existence of other IPRs not referenced in the present document which are, or may be, or may become, essential to the present document.

Legal notices

The Cloud Signature Consortium seeks to promote and encourage broad and open industry adoption of its standard.

The present document does not create legal rights and does not imply that intellectual property rights are transferred to the recipient or other third parties. The adoption of the specification contained herein does not constitute any rights of affiliation or membership to the Cloud Signature Consortium VZW.

This document is provided “as is” and the Cloud Signature Consortium, its members and the individual contributors, are not responsible for any errors or omissions.

1 Scope

When digital signatures are created within a device, the interfaces and functions are standardized, e.g. the API used by the application program to access the signature creation libraries and the interface to the smart card or similar device (if a device is used) holding the signing key. When digital signatures move to the cloud, the functions needed to create a digital signature can be distributed across several service instances, each carrying out one or more steps in the signature creation process. The interfaces between such services are however until now not standardized.

The Cloud Signature Consortium aims to fill this gap in standardization by defining the architectural design, communication protocols, application programming interfaces, data structures, and technical requirements needed to establish interoperable solutions for cloud-based digital signatures. While these specifications are applicable in a wide variety of use cases with different security requirements, the fulfilment of requirements imposed by the eIDAS Regulation of the EU [i.1] is particularly addressed, supporting the creation of “advanced” or “qualified” electronic signatures and electronic seals in the cloud.

This document contains technical specifications that are intended for use by applications for creating digital signatures in the cloud and by a variety of applications consuming these services. By implementing their services according to these specifications, service providers can ensure that services are applicable as parts of complete digital signature systems in the cloud in a plug and play manner.

Existing standards and open specifications are considered by the consortium as far as applicable.

The following are out of scope of this specification:

- Policy requirements for (qualified and other) service providers; this is an area of standardization covered by ETSI.
- Signing key creation and enrollment; although keys MAY be created by the remote service during the signing workflow, these activities are not covered by specific API methods.
- Signature and certificate formats; use of the standards specified by ETSI is RECOMMENDED.
- Signature validation; this will be addressed in future specifications from the Consortium.
- Security evaluation and requirements for hardware components used to hold signing keys (HSM – hardware security module); this is being standardized by CEN in Europe and FIPS in the USA.
- Internal functionality and internal interfaces in service provider systems.

Note that the current specifications mainly cover architectures where the signing key is held “in the cloud”, i.e. by a signature creation device managed by a service provider. Architectures where the signing key is in the hand of the signer, stored in the user’s device or in an attached smart card or similar, are not covered as a particular case. The consortium will consider the need for further specifications covering situations where a user device holding the signing key interacts with cloud

services for digital signature creation, e.g. cloud services MAY be used for document storage, hash computation, and signature formatting.

2 Interpretation of Requirement Levels

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [1].

3 References

3.1 Normative references

The following documents, in whole or in part, are normatively referenced in this specification and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or errata) applies.

[1] IETF RFC 2119: “Key words for use in RFCs to Indicate Requirement Levels”.

[2] IETF RFC 3161: “Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)”.

[3] IETF RFC 3986: “Uniform Resource Identifier (URI): Generic Syntax”.

[4] IETF RFC 4514: “Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names”.

[5] IETF RFC 4627: “The application/json Media Type for JavaScript Object Notation (JSON)”.

[6] IETF RFC 4648: “The Base16, Base32, and Base64 Data Encodings”.

[7] IETF RFC 5246: “The Transport Layer Security (TLS) Protocol Version 1.2”.

[8] IETF RFC 5280: “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”.

[9] IETF RFC 5646: “Tags for Identifying Languages”.

[10] IETF RFC 5816: “ESSCertIDv2 Update for RFC 3161”.

[11] IETF RFC 6749: “The OAuth 2.0 Authorization Framework”.

[12] IETF RFC 6750: “The OAuth 2.0 Authorization Framework: Bearer Token Usage”.

[13] IETF RFC 7009: “OAuth 2.0 Token Revocation”.

[14] IETF RFC 7235: “Hypertext Transfer Protocol (HTTP/1.1): Authentication”.

[15] IETF RFC 7518: “JSON Web Algorithms (JWA)”.

[16] IETF RFC 7519: “JSON Web Token (JWT)”.

[17] IETF RFC 7521: “Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants”

[18] IETF RFC 8017: “PKCS #1: RSA Cryptography Specifications Version 2.2”.

[19] IETF RFC 8446: “The Transport Layer Security (TLS) Protocol Version 1.3”.

[20] IETF draft-ietf-oauth-security-topics: “OAuth 2.0 Security Best Current Practice”

[21] ETSI TS 119 312: “Electronic Signatures and Infrastructures (ESI); Cryptographic Suites”.

[22] ISO 3166-1: ” Codes for the representation of names of countries and their subdivisions — Part 1: Country codes”.

[23] IETF RFC 8414: “OAuth 2.0 Authorization Server Metadata”

[24] IETF RFC 7591: “OAuth 2.0 Dynamic Client Registration Protocol”

[25] IETF RFC 7636: “Proof Key for Code Exchange by OAuth Public Clients”

[26] IETF RFC 8705: “OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens”

[27] IETF Draft draft-ietf-oauth-rar: “OAuth 2.0 Rich Authorization Requests”

[28] IETF Draft draft-ietf-oauth-par: “OAuth 2.0 Pushed Authorization Requests”

[29] ETSI EN 319 122-1 “Electronic Signatures and Infrastructures (ESI); CAdES digital signatures; Part 1: Building blocks and CAdES baseline signatures” :::

[30] ETSI EN 319 132-1: “Electronic Signatures and Infrastructures (ESI); XAdES digital signatures; Part 1: Building blocks and XAdES baseline signatures”

[31] ETSI EN 319 142-1: “Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 1: Building blocks and PAdES baseline signatures”

[32] ETSI TS 119 182-1: “Electronic Signatures and Infrastructures (ESI); JAdES digital signatures; Part 1: Building blocks and JAdES baseline signatures”

[33] IETF RFC 6960: “X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP”

3.2 Informative references

The following documents, in whole or in part, are informatively referenced in this specification and may be a useful contribution for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or errata) applies.

[i.1] Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the

internal market and repealing Directive 1999/93/EC.

[i.2] ETSI SR 019 020: “The framework for standardization of signatures; Standards for AdES digital signatures in mobile and distributed environment”.

[i.3] IETF RFC 3447: “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1”.

[i.4] IETF RFC 6101: “The Secure Sockets Layer (SSL) Protocol Version 3.0”.

[i.5] CEN EN 419 241-1: “Trustworthy Systems Supporting Server Signing - Part 1: General System Security Requirements”

[i.6] ISO/IEC 19790: “Information technology - Security techniques - Security requirements for cryptographic modules”

[i.7] Hickman, Kipp, “The SSL Protocol”, Netscape Communications Corp., Feb 9, 1995

[i.8] ETSI TS 119 001: “Electronic Signatures and Infrastructures (ESI); The framework for standardization of signatures; Definitions and abbreviations.

[i.9] ETSI TS 119 312: “Electronic Signatures and Infrastructures (ESI); Cryptographic Suites.

[i.10] South African Act No. 25 of 30 August 2002: Electronic Communications and Transactions Act, 2002

[i.11] Web Authentication: An API for accessing Public Key Credentials Level 2, 2021

[i.12] OpenID Connect Core 1.0 incorporating errata set 1, 2014

4 Terms, definitions and abbreviations

4.1 Terms and definitions

For the purposes of this specification, the following terms and definitions apply.

access token: credentials used to access protected resources. It’s a string representing an authorization issued to the client. The string is usually opaque to the client.

Note 1: As defined in IETF RFC 6749 [11].

authentication factor: piece of information and/or process used to authenticate or verify the identity of an entity.

Note 2: As defined in ISO/IEC 19790 [i.6].

EXAMPLE: A password or PIN.

authorization server: The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

Note 3: As defined in IETF RFC 6749 [11].

credential: cryptographic object and related data used to support remote digital signatures over the Internet. Consists of the combination of a public/private key pair (also named “signing key” in CEN EN 419 241-1 [i.5]) and a X.509 public key certificate managed by a remote signing service provider on behalf of a user.

digital signature: data appended to, or a cryptographic transformation (see cryptography) of a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery e.g. by the recipient [i.8]

Note 4: Digital signature is a technical term. Specifically, but not exclusively, aims at supporting legal terms as electronic signatures, advanced electronic signatures, qualified electronic signatures, electronic seals, advanced electronic seals, and qualified electronic seals as per Regulation (EU) No 910/2014 [i.1].

electronic signature: digital signature created by using a certificate issued to a natural person ensuring the integrity and origin of the document and the signatory commitment to the document content. ::: {.NOTE} Electronic signatures used in the present document are meant to specifically, but not exclusively, support the electronic signatures defined as per Regulation (EU) No 910/2014 [i.1]. :::

electronic seal: digital signature created by using a certificate issued to a legal person or business unit ensuring the integrity and origin of the document, without necessarily committing to the content.

Note 5: Electronic seals used in the present document are meant to specifically, but not exclusively, support the electronic seals defined as per Regulation (EU) No 910/2014 [i.1].

identity proofing: process by which the identity of an applicant is verified by the use of evidence attesting to the required identity attributes

Note 6: Depending on how the claims will be used, different assurance levels will be required when verifying the claims.

remote service: service implementing the API described in this specification and delivered on the Internet.

remote signing service provider: service provider managing a set of credentials on behalf of multiple users and allowing them to create a remote signature with a stored credential.

Note 7: A remote signing service provider typically operates an HSM (or functionally equivalent multi-user secure device) and an authentication service. It manages the users and provides a signing service that can be accessed over the Internet by means of the API described in this specification.

Note 8: A remote signing service typically manages signing keys and certificates that are created before the signing operations take place. Another common scenario is when the signing key and the certificate are created in the course of a signing operation. In the present specification, this

is referred to as “Short-Lived Credential Signing” (also called “ad-hoc” or “on-the-go” credential signing).

remote signature creation device: signature creation device used remotely from signer perspective to provide control of signing operation on its behalf of the signer.

short lived credentials: temporary credentials created to sign a specific transaction where those credentials will then expire or be explicitly revoked shortly after being applied in the signature operation. Methods to create and manage short lived credentials across multiple transactions will be handled in a future release of this specification.

::: {.NOTE} Once the end-user has had their claims successfully verified in an identity proofing process, they become eligible to sign with short-lived credentials. The assurance level of the claims associated with the identity will determine the trust level that can be achieved with the short-lived credentials.

signature activation data: set of data used to control a given signature operation, performed by a cryptographic module, on behalf of the signer.

signature activation module: configured software that uses the SAD in order that the signing keys are used under sole control of the signer.

Note 9: As defined in CEN EN 419 241-1 [i.5].

signature application: client application or service calling the remote signing service provider to create a remote signature.

signature application provider: service provider managing a signature application and offering it as a service over the Internet or other communication channel.

4.2 Abbreviations

AdES: Advanced Electronic Signature

API: application programming interface

HSM: hardware security module

RSCD: remote signature creation device

RSSP: remote signing service provider

SAD: signature activation data

SAM: signature activation module

SCAL1: sole control assurance level 1

Note 10: As defined in CEN EN 419 241-1 [i.5].

SCAL2: sole control assurance level 2

Note 11: As defined in CEN EN 419 241-1 [i.5].

SDR: signer's document representation

5 Conventions

5.1 Text conventions

This specification adopts the following text conventions to help identify various types of information.

Table 1 – Text conventions

Text convention	Example
The vertical bar (|) indicates a possible value for selection or outcome and SHALL be interpreted as "or".	YES | NO
Text in colored boxes is example code.	`POST /csc/v2/credentials/info HTTP/1.1`
Bold text indicates the name of an API method.	**credentials/list**
Italic text indicates the name of an API input or output parameter.	*access_token*

In general, API names as well as API input or output parameters defined in this specification use the "camelCase" notation, like *authType* or **credentials/extendTransaction**. However, names and parameters that are defined in other standards, like those in the domain of authentication and related to OAuth 2.0, are used here in their original format to facilitate understanding and interoperability, using "snake_case", like *refresh_token*, i.e., two names separated by an underscore.

5.2 Base64

When data is required to be Base64-encoded, it SHALL be encoded as "base64" as defined in RFC 4648 [6]. To avoid JSON representation issues line breaks SHALL NOT be used within Base64-encoded data. When data is base64url-encoded it SHALL be encoded as "base64url" as defined in RFC 4648 [6].

6 Architectures and use cases

The present specification and the protocols defined herein aim to support different use cases. However, they focus on the scenario of remote signing defined for example as "the creation of remote electronic signatures, where the electronic signature creation environment is managed by a trust service provider on behalf of the signatory" in EU Regulation 910/2014 [i.1], whereas §52.

This means that other scenarios for signing in distributed environments assisted by remote servers – like those described in ETSI SR 019 020 [i.2]("Standards for AdES digital signatures in mobile and distributed environment") – are not covered in the present version of this specification. In particular, use cases where the signing key is contained within a signer's personal device are not covered: for example, signing a document located on a server with a private key contained in a mobile SIM card,

or in a cryptographic device connected to a personal computer. These are relevant use cases, although not fitting in the core definition of “remote signature”, so they may be specifically covered in future updates of the specification.

6.1 Supported architectures

The current version of the specifications focuses on the interface between the Signature Application and the remote signing service provider. The following figure shows a typical but not restrictive example of the architecture.

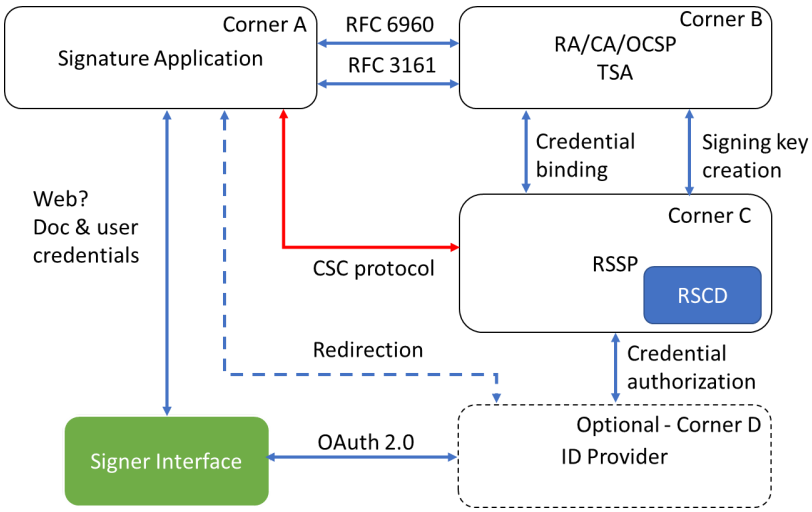

Figure 1: Remote signing corners

There are four main corners in the remote signing scenario.

The Signature Application retrieves the document to be signed from the user, and, if needed the certificates, revocation information and time-stamps from the corresponding trust service provider. It requests the remote signing service provider to create the signature of the hash value.

The RSSP connects to the CA for the credential binding. In some cases, the CA may also be included in the process of creating the signing key.

The authorization for service or credential access can be done either passing through the signature application or using a redirection to an external OAuth 2.0 authorization server (AS). In many cases, the authorization server is part of the RSSP.

The redirect-based model employed by OAuth allows the RSSP to utilize FIDO/WebAuth [i.11] or 3rd party identity providers (e.g. via OpenID Connect [i.12]) for user authentication.

7 Introduction to the remote service protocols API

Web applications and services use Application Programming Interfaces (APIs) to talk to each other. Technically speaking, in the web service context, an API is a set of programming instructions for accessing a Web-based software application or service.

The remote service protocols API allows a signature application to communicate with a remote service via the Internet by leveraging a sequence of calls to methods.

7.1 Format and syntax of the API

This specification defines Web services APIs that are based on technical standards and protocols such as HTTP and JSON. This API uses HTTP POST requests with JSON payload and JSON responses. JSON is an open-standard media type format as defined by RFC 4627 [5] that uses human-readable text to transmit data objects consisting of attribute-value pairs. These properties make JSON an ideal data-interchange language which is used as the most common data format for asynchronous communications.

The functions offered by the remote service are represented by HTTP RPC endpoints accepting arguments as JSON in the request body and returning results as JSON in the response body. For this reason, the HTTP header of the invocation method SHALL include a Content-Type: application/json header.

The remote service SHALL use HTTP version 1.1 or higher.

A JSON schema corresponding to the API defined in the present specification is available. See JSON schema and OpenAPI description.

7.2 Remote service base URI

The remote service base URI defines the style and format of the HTTP endpoint URI of a remote service conforming to this specification.

The base URI contains the version number of the APIs that is implemented by the remote signing service provider. In the case of this specification, the version number SHALL be `v2`. Future versions of this specification MAY not be completely backward compatible.

```
https://service.domain.org/xxx/csc/v2/
```

The base URI SHALL start with an arbitrary URL defined by the service provider (‘https://service.domain.org/xxx’ in the example above) and SHALL end with ‘/csc/v2’. The endpoints of the API methods documented in this specification SHALL be concatenated to the base URI. An exception is given by the OAuth 2.0 methods, as defined in OAuth 2.0 Authorization, which MAY use URIs that are independent from the service base URI.

7.3 Integrity and confidentiality

A remote service conforming to this specification SHALL guarantee the integrity and confidentiality of the communication channel between the signature application and the remote service.

The integrity and confidentiality of the communication channel between the user and the signature application or the remote service are out of the scope of this specification.

The remote service SHOULD implement Transport Layer Security (TLS) in order to ensure the integrity and confidentiality of the communications. This prevents easy eavesdropping or impersonation if authentication credentials are hijacked. Another advantage of always using TLS is that guaranteed encrypted communications simplifies the authentication schemes, so for example

simple mechanisms like Basic HTTP authentication can be used because the elements used in the authentication (username and password) are always transmitted over an encrypted channel.

The remote service MAY use other methods than TLS, for example using VPN.

TLS 1.3 as described in RFC 8446 [19] is, at the time of this writing, the latest version of TLS. Until TLS 1.3 is widely adopted, the previous version TLS 1.2 as described in RFC 5246 [7] SHALL be supported by remote services conforming to this specification and is the RECOMMENDED mechanism to use for interoperability reasons. TLS 1.2 provides access to advanced cipher suites that support elliptic curve cryptography and authenticated encryption with associated data (AEAD) block cipher modes. TLS 1.1 MAY be used, but it is also less secure. TLS 1.0 is considerably less secure and some security certifications like PCI DSS 3.1 explicitly forbid it, so remote services SHOULD NOT support it.

All versions of SSL (SSLv3 as defined in RFC 6101 [i.4] or SSLv2 as defined in [i.7]), the security protocol used before TLS, are considered insecure. Remote services conforming to this specification SHALL NOT implement SSL.

7.4 Remote service information

This specification defines a protocol to connect a signature application to a remote service. Other similar specifications exist in the industry, but they are typically proprietary and incompatible between each other, so if a signature application wants to support multiple remote services, then the development effort would increase significantly.

This specification has been designed to support modular services that may be implemented in line with the capacity and mission of the provider. This means that a remote service that supports this specification MAY implement only a subset of the API methods defined herein. In order to facilitate this approach, this specification defines the **info** method, which all remote services SHALL implement to allow the signature application to discover which of the API methods are supported.

In addition, the **info** method returns information on the remote service which may be useful to a calling application to access the functions and features of the service.

7.5 *clientData* parameter

Most methods allow to provide *clientData* as an optional input parameter. It can contain any arbitrary data from the signature application. This data allows the signature application to handle other application-specific data like, e.g., a transaction identifier.

The remote service MAY use this information and it MAY also log this data together with information of the call. This parameter MAY expose sensitive data to the remote service. Therefore, it SHOULD be used carefully by signature applications.

7.6 Expressing algorithms

The present document expresses algorithms via Object IDentifiers (OID). OIDs are identifiers standardized by the Internal Telecommunication Union (ITU) and ISO/IEC to identify a specific object. They are represented by numbers, separated by dots, and are constructed in a tree-like structure. A list of the most common OIDs for algorithms used in signatures can be found in chapter 10 of ETSI TS 119 312 [i.9]. See also the OID repository http://oid-info.com in search of specific OIDs.

8 Authentication and authorization

This specification supports two types of authentication and authorization:

a. Service authorization and authentication.

b. Credential authorization.

8.1 Service authorization and authentication

In order to protect the remote service from unauthorized access, this specification requires the signature application to obtain a valid "access token" to authorize the access to the APIs. This type of authorization is called service authorization. Various types of authorization mechanisms can be supported, and more will be supported in future versions, and the signature application SHALL adopt any of those available from the remote service as stated in the response to the **info** method, as defined in info.

The remote service MAY also adopt an indirect way of authorizing access to the API. The underlying communication channel with the signature application MAY ensure access control in a different way, for example with a private point-to-point LAN connection or through a VPN (Virtual Private Network).

The access to the APIs SHALL be authenticated. When the authentication is under the control of the signature application provider, then the user SHALL be properly authenticated by this provider before getting access to the remote service. This scenario supports organizations that manage a user community with an existing form of authentication, for example a Bank managing the users from their Internet Banking service. This means that, in order to retrieve the signing credentials associated to a user, this organization would have to take care of the correspondence between the user identifier in their own domain and the user identifier in the remote service's domain.

When the authentication is under the control of the remote service, the signature application SHALL perform a token-based authentication to the remote service by means of authentication factors collected from the user, preferably via an OAuth 2.0 authorization mechanism, or through HTTP Basic or HTTP Digest authentication. In case the signature application is not under the control of the user, OAuth 2.0 authorization SHOULD be used. In practice, the signature application will require the user to authenticate directly to the remote service using any of the available methods. This would offer an authentication mechanism even in case the signature application and the remote service have not previously established any form of service authentication.

Two methods are defined in this specification to obtain an access token to authorize the access to the remote service API:

- The **oauth2/token** method SHALL be used when an OAuth 2.0 authorization mechanism is supported by the remote service. The signature application will not collect any authentication factors from the user, but instead it will redirect to the remote service that will authenticate the user. See OAuth 2.0 Authorization for further information on how to implement OAuth 2.0 authorization.

- The **auth/login** method SHALL be used when OAuth 2.0 is not available and HTTP Basic or Digest authentication mechanisms are preferred and supported by the remote service. The

signature application will collect the authentication factors from the user and will submit them to the remote service to obtain an authorization.

In both cases, if the user grants the authorization, the remote service will return a service access token to the signature application. From then on, all authenticated requests to the API methods defined in this specification SHALL use an Authorization header with *Bearer* type followed by that service access token.

If the user does not grant the authorization, the authorization server will return an error message and no access to authenticated API methods will be possible.

8.2 Credential authorization

Accessing a credential for remote signing requires an authorization from the user who owns the signing key associated to it. As a special case, the user might also authorize the creation of one or more signatures along with a signature qualifier instead of a particular credential identification. This is especially useful in conjunction with short-lived credentials.

The remote service can manage the authorization in multiple ways, with different technologies and a variable number of authorization factors. This really depends on the implementation and on the policy adopted by the remote service, and MAY also be determined by the level of compliance to industry and regulatory requirements, like in the case of standards like CEN EN 419 241-1 [i.5], which defines different “sole control assurance levels”, SCAL1 and SCAL2.

For a precise description of the difference between SCAL1 and SCAL2 we refer to CEN EN 419 241-1 [i.5]. However, with regards to this specification, two aspects should be noted about SCAL2:

1. The signature activation data, used to authorize a signature, is linked to the document or the documents to be signed.

2. A two-factor authorization is needed to authorize a signature.

Two different types of credential authorization are defined and supported in this specification:

- Explicit authorization

- OAuth 2.0 authorization

Explicit authorization means that the remote service relies on the signature application to collect, in its own environment, authentication factors like PIN or One-Time Passwords (OTP), according to the parameters returned by the **credentials/info** method, as defined in credentials/info. This method returns the type, format and combination of required or optional authentication factors, such that the signature application could show the proper interactive controls to collect them from the user.

A common type of explicit authorization is based on a static PIN - typically defined by the user - associated to the signing key when it is generated. To increase the level of assurance of user control, ensuring that only the authorized user could create a signature with a certain credential, a stronger authorization factor MAY be adopted. A dynamically generated text-based One-Time Password (OTP) is a common strong authorization mechanism. This specification directly supports the combination of various mechanisms which can be used complementary to service authorization to achieve the highest levels of assurance of the user’s sole control, and can be used to support SCAL1 and SCAL2 as defined in CEN 419 241-1 [i.5].

Biometric authentication and phone call drop are other examples of possible authorization mechanisms. As these and other authorization mechanisms require a very peculiar user interface, they can be supported by means of an OAuth 2.0-based authorization scheme.

8.3 Explicit credential authorization

To be able to support the broadest range of authorization mechanisms, this specification provides a generic way to define access control to credentials. Each credential is associated with a set of **authentication object types** and an access rule describing the precondition to authorize the credential access.

8.3.1 Authentication objects

An **authentication object** type describes the data structure and protocol of authentication mechanisms, much the same way as it is done in the PKCS#15 standard. Authentication object types are returned by the *credentials/info* method, such that the Signature Application can show the proper interactive controls to collect them from the user.

Each authentication object type is associated with a `type` property, defining both the data structure that a client application SHALL provide and the protocol that SHALL be processed. The `id` property is used to identify the associated authentication object type in a concrete authentication object data structure.

The number and type of authentication object types is provider specific.

Depending on the authentication object type the Signature Application collects concrete authentication object data and drives the associated protocol. The authentication object data is sent using the method *credentials/authorize*.

The following is an example authentication object type, describing the need for a password entry:

```
{
    "type": "Password",
    "id": "PIN",
    "label": "Personal PIN",
    "format" : "A"
}
```

This indicates to the client that it needs to send an alphanumeric password within a later authorization request, identifying it as “PIN”. When requesting user input, the client may present the required data as “Personal PIN” to the user.

In consequence, the client might send an authentication object as seen in the following example:

```
{
    "id": "PIN",
    "value": "1234"
}
```

This example assumes that the client has received a PIN value of “1234”, which is conveyed to the authorization endpoint.

See the following sections for a thorough description of these data structures.

8.3.1.1 Out-of-band response

“Out-of-band response” is used here whenever an authentication object is sent to the service provider by using some protocol and session not associated and described in this API specification. This can be for example a SMS, phone or email channel.

With an out-of-band response the call to *credentials/authorize* does not have any knowledge about the state of the out-of-band task. Processing of the call can be implemented using a polling or blocking approach. As such, the processing can either

- terminate with a HTTP 200, returning the specified result tokens.
- terminate with a HTTP 202 as an indication that the out-of-band result is not yet available. The client has to re-issue a request to *credentials/authorizeCheck* (polling). Eventually the request will terminate with an error or HTTP 200.

8.3.1.2 Common properties

The following properties are common to all authentication object types as they are received from *credentials/info*.

Name	Presence	Description
type	REQUIRED	The type of the authentication object. This describes the data structure and protocol. The value SHALL be one of the tokens defined in this specification. A provider MAY not support all token types.
id	REQUIRED	The unique identifier of the authentication object.
label	OPTIONAL	A label to be presented to the user. It is used to identify the requested authentication data in human readable manner.
description	OPTIONAL	A description to be presented to the user. It carries instructions on how to provide the authentication data.

The following properties are common to all authentication objects as they are sent via *credentials/authorize*.

Name	Presence	Description
id	REQUIRED	The unique identifier of the authentication object.

8.3.1.3 Password, in band response

The **Password** type simply requires the client to collect authentication information from the user and send it to the provider in-band.

Be aware that from a provider point of view an OTP generated statically / offline by a client side token is simply a “Password” type, too.

This authentication object type allows for the definition of - Simple password authentication - “offline” OTP generation - Combinations thereof, e.g. the requirement of having two PINs entered (4 eyes).

Authentication type properties:

Name	Presence	Value	Description

Name	Presence	Value	Description
type	REQUIRED	“Password”	
format	OPTIONAL	“A”|“N”	Specifies the format of the password: - “A”: alphanumeric text; allowed characters: A-Z	a-z	0-9 - “N”: numeric text If omitted, any character is allowed.
generator	OPTIONAL	String	If a client side device or algorithm is needed to derive the password, it can be referenced by this property. E.g. a trust service provider can have issued multiple tokens and allows the user to identify the required one using this property.

Authentication object properties:

Name	Presence	Description
value	REQUIRED	The concrete password value.

Example I authentication type:

```
{
    "type": "Password",
    "id": "PIN",
    "label": "PIN",
    "format" : "N"
}
```

Example I authentication object:

```
{
    "id": "PIN",
    "value": "1234"
}
```

Example II authentication type:

```
{
    "type": "Password",
    "id": "OTP",
    "label": "OTP",
    "generator" : "b23",
    "format" : "A"
}
```

Example II authentication object:

```
{
    "id": "OTP",
    "value": "3rfd45s"
}
```

8.3.1.4 Password, out of band response

The **PasswordOOB** indicates that by some unspecified mechanism an authentication object is sent to the service provider.

This authentication object type allows for the definition of - SMS, phone or email authorization - Provider-specific authorization without user agent intervention

Authentication type properties: