Signature Validation Token

Version 1.0 - 2020-03-12 - *Draft version*

Registration number: **2020-60**

Copyright © The Swedish Agency for Digital Government (DIGG), 2015-2020. All Rights Reserved.

Table of Contents

1. Introduction

Electronic signatures have a limited lifespan regarding when they can be validated and determined to be authentic. Many factors make it more difficult to validate electronic signatures over time. For example:

- Trusted information about the validity of the signing certificate is not available.
- No proof of time when the signature was actually created is available.
- Algorithms used to create the signature is no longer considered secure.
- Services necessary to validate the signature are no longer available.
- Inability to verify supporting evidence such as, CA certificates, OCSP responses, revocation lists or timestamps.

The challenge to validate an electronic signature is increasing over time up until the point when it is simply impossible to verify the signature with a sufficient level of assurance.

Current existing standards such as the ETSI AdES profiles for CMS, XML and PDF signatures can be used to prolong the lifetime of a signature by storing data that supports validation of the signature beyond the lifetime of the signing certificate. The problem with this approach is that the amount of information that must be stored along with the signature is constantly growing over time. The increasing amount of information and signed objects that must be validated in order to verify the original signature is growing in complexity to the point where it may become infeasible to validate the original signature.

The Signature Validation Token (SVT) defined in this specification takes a fundamentally different approach to the problem by providing an evidence that asserts the validity of a signature. The SVT is issued by a trusted authority, and asserts that a particular signature was successfully validated according to defined procedures at a certain time. The basic idea and intent behind the SVT is that once the SVT is issued by a trusted authority, any future validation of that signature is satisfied by validating the SVT without any need to also validate the original signature.

This approach drastically reduces the complexity of signature validation of older signatures for the simple reason that validating the SVT just requires validation of the signature over the SVT. The SVT can be signed with keys and algorithms that makes it valid for a considerable time in the future and it can be re-issued with fresh keys and signatures to extend the lifetime of the original signature validity, if necessary.

1.1. Requirements Notation

The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL** in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

These keywords are capitalized when used to unambiguously specify requirements over protocol features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

1.2. Definitions

This document use the following defined terms:

Term	Meaning
Signed Data	The data covered by a particular signature. This is typically equivalent to the signed document and represents the data that the signer intended to sign. In some cases, such as in some XML signatures, the signed data can be the collection of several data fragments each referenced by the signature. In the case of PDF, this is the data covered by the "ByteRange" parameter in the signature dictionary.

Term	Meaning
Signed Bytes	These are the actual bytes of data that was hashed and signed by the signature algorithm. In most cases this is not the actual Signed Data, but a collection of signature metadata that includes references (hash) of the Signed Data as well as information about algorithms and other data bound to a signature. In XML this is the canonicalized SignedInfo element and in CMS/PDF signatures this is the DER encoded SignedAttributes structure.

When these terms are used in their defined meaning, they appear with a capitalized first letter as shown in the table.

2. Signature Validation Token

2.1. Function

The function of the Signature Validation Token (SVT) is to capture evidence of signature validity at one instance of secure signature validation process and to use that evidence to eliminate the need to perform any repeated cryptographic validation of the original signature value, as well as reliance on any hash values bound to that signature. The SVT achieves this by binding the following information to a specific electronic signature:

- A unique identification of the signature.
- The data and metadata signed by the signature.
- The signer's certificate that was validated as part of signature validation.
- The certificate chain that was used to validate the signer's certificate.
- An assertion providing evidence of that the signature was validated, when in time the validation was performed, which procedures that was used to validate the signature, and the outcome of the validation.
- An assertion providing evidence of the point in time at which the signature is known to have existed, which procedures that was used to validate the time of existence and the outcome of the validation.

Using an SVT is equivalent to validating a signed document in a system once and then using that document multiple times without revalidating the signature for each usage. Such procedures are common in systems where the document is residing in a safe and trusted environment where it is protected against modification. The SVT allows the time and environment where the document can be stored and used to expand beyond a locally controlled environment and a short instance of time.

Using the SVT, the signed document can be validated once using a reliable trusted service and after that the SVT can be used to extend reliance of that secure validation process. The SVT is therefore not only a valuable tool to extend the lifetime of a signed document, but also useful since a single secure validation service can be deployed, instead of having to maintain close integrations between signature validations and document usage.

2.2. Signature Validation Token Syntax

The SVT is carried in a JSON Web Token (JWT) in accordance with [RFC7519].

2.2.1. Data Types

The contents of claims in an SVT are specified using the claims data types in the following table:

Claims Data Type	JSON Data Type	Description
String	string	An arbitrary case sensitive string value.
Base64Binary	string	String representation of Base64 encoded byte array of binary data.
StringOrURI	string	As defined in [RFC7519]. A JSON string value, with the additional requirement that while arbitrary string values MAY be used, any value containing a ":" character MUST be a URI.
URI	string	A valid URI.
Integer	number	A 32-bit signed integer value (from -2^{31} to $2^{31} - 1$).

Claims Data Type	JSON Data Type	Description
Long	number	A 64-bit signed integer value (from -2^{63} to $2^{63} - 1$).
NumericDate	number	As defined in [RFC7519]. A JSON numeric value representing the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds.
Boolean	boolean	The explicit value true or false.
Object<Class>	object	A JSON object holding a claims object of a class defined in this specification (see section 2.2.2).
Map<Type>	object	A JSON object with name-value pairs where the value is an object of the specified Type in the notation. Example: Map<String> according to this notation is a JSON object with name value pairs where all values are of type String.
Array	array	An array of values of a specific data type as defined in this table. An array is expressed in this specification by square brackets. For example: **[String]** indicates an array of String values and **[Object<DocHash>]** indicates an array of DocHash objects.
Null	null	Representing an absent value. A claim with a null value is equivalent with an absent claim in this specification.

2.2.2. Signature Validation Token JWT Claims

The signature validation token JWT SHALL contain claims according to the following table.

Name	Data Type	Value	Presence
`jti`	**String**	A "JWT ID" registered claim according to [RFC7519]. It is RECOMMENDED that the identifier holds a hexadecimal string representation of a 128-bit unsigned integer.	MANDATORY
`iss`	**StringOrURI**	An "Issuer" registered claim according to [RFC7519]. An arbitrary unique identifier of the SVT issuer. This value SHOULD have the value of an URI identifier based on a domain owned by the issuer.	MANDATORY
`iat`	**NumericDate**	An "Issued At" registered claim according to [RFC7519] expressing the time when this SVT was issued.	MANDATORY
`aud`	**[StringOrURI]** or **StringOrURI**	An "Audience" registered claim according to [RFC7519]. The audience claim is an array of one or more identifiers, identifying intended recipients of the SVT. Each identifier MAY identify a single entity, a group of entities or a common policy adopted by a group of entities. If only one value is provided it MAY be provided as a single **StringOrURI** value instead of as an array of values.	OPTIONAL

Name	Data Type	Value	Presence
`exp`	**NumericDate**	An "Expiration Time" registered claim according to [RFC7519] expressing the time when services and responsibilities related to this SVT is no longer provided by the SVT issuer. The precise meaning of the expiration time claim is defined by local policies. See implementation note below.	OPTIONAL
`sig_val_claims`	**Object<SigValidation>**	Signature validation claims for this SVT extending the standard registered JTW claims above.	MANDATORY

> **Note**: An SVT asserts that a certain validation process was undertaken at a certain instance of time. This fact never changes and never expires. However, some aspects of the SVT such as liability for false claims or service provision related to a specific SVT may expire after a certain period of time, such as a service where an old SVT can be upgraded to a new SVT signed with fresh keys and algorithms.

2.2.3. Claim Object Classes

2.2.3.1. The SigValidation Object

The **SigValidation** claims object holds all custom claims of the SVT JWT and contains the following parameters:

Name	Data Type	Value	Presence
`ver`	**String**	Version. This version is indicated by the value "1.0".	MANDATORY
`profile`	**StringOrURI**	Name of a profile applied to this specification that defines conventions of content of specific claims and extension points.	OPTIONAL
`hash_algo`	**URI**	The URI identifier of the hash algorithm used to provide hash values within the SVT. The URI identifier SHALL be one defined in [RFC6931] or in the IANA registry defined by this RFC.	MANDATORY
`sig`	**[Object<Signature>]**	Information about validated signatures as an array of **Signature** objects. If the SVT contains signature validation evidence for more than one signature, then each signature is represented by a separate **Signature** object. At least one **Signature** object MUST be present.	MANDATORY
`ext`	**Map<String>**	Extension point for additional claims related to the SVT. Extension claims are added at the discretion of the SVT issuer but MUST follow any conventions defined in a profile of this specification (see section 3).	OPTIONAL

2.2.3.2. The Signature Claims Object

The **Signature** object contains claims related to signature validation evidence for one signature and contains the following parameters:

Name	Data Type	Value	Presence

Name	Data Type	Value	Presence
`sig_ref`	**Object<SigReference>**	Reference information identifying the target signature.	MANDATORY
`sig_data`	**[Object<SignedData>]**	Array of references to Signed Data signed by the target signature.	MANDATORY
`signer_cert_ref`	**Object<CertReference>**	Reference to signer certificate and optionally reference to a supporting certificate chain that was used to validate the target signature.	MANDATORY
`sig_val`	**[Object<PolicyValidation>]**	Array of results of signature validation according to defined validation procedures.	MANDATORY
`time_val`	**[Object<TimeValidation>]**	Array of results of time verification validating proof that the target signature has existed at specific instances of time in the past.	OPTIONAL
`ext`	**MAP<String>**	Extension point for additional claims related to the target signature. Extension claims are added at the discretion of the SVT Issuer but MUST follow any conventions defined in a profile of this specification (see section 3).	OPTIONAL

2.2.3.3. The SigReference Claims Object

The **SigReference** claims object provides information used to match the **Signature** claims object to a specific target signature and to verify the integrity of the target signature value and Signed Bytes.

Name	Data Type	Value	Presence
`id`	**String**	Optional identifier assigned to the target signature.	OPTIONAL
`sig_hash`	**Base64Binary**	Hash value of the target signature value.	MANDATORY
`sb_hash`	**Base64Binary**	Hash value of the Signed Bytes of the target signature.	MANDATORY

2.2.3.4. The SignedData Claims Object

The **SignedData** claims object provides information used to verify the target signature references to Signed Data as well as to verify the integrity of all data signed by the target signature.

Name	Data Type	Value	Presence
`ref`	**String**	Reference identifier identifying the data or data fragment covered by the target signature.	MANDATORY
`hash`	**Base64Binary**	Hash of the data covered by the target signature.	MANDATORY

2.2.3.5. The PolicyValidation Claims Object

The **PolicyValidation** claims object provide information about the result of a validation process according to a spefific policy.

Name	Data Type	Value	Presence
`pol`	**StringOrURI**	Identifier of the policy governing the validation process.	MANDATORY
`res`	**String**	Result of the validation process. The value MUST be one of "PASSED", "FAILED" or "INDETERMINATE" as defined by [ETSI EN 319 102-1].	MANDATORY
`msg`	**String**	An optional message describing the result.	OPTIONAL
`ext`	**Map<String>**	Extension for additional information about the validation result.	OPTIONAL

2.2.3.6. The TimeVerification Claims Object

The **TimeVerification** claims object provide information about the result of validating time evidence asserting that the target signature existed at a particular time in the past.

Name	Data Type	Value	Presence
`time`	**NumericDate**	The verified time.	MANDATORY
`type`	**StringOrURI**	Identifier of the type of evidence of time.	MANDATORY
`iss`	**StringOrURI**	Identifier of the entity that issued the evidence of time.	MANDATORY
`id`	**String**	Unique identifier assigned to the evidence of time.	OPTIONAL
`val`	**[Object<PolicyValidation>]**	Array of results of validation of the time evidence according to defined validation procedures.	OPTIONAL
`ext`	**Map<String>**	Extension for additional information about the signature validation result.	OPTIONAL

2.2.3.7. The CertReference Claims Object

The **CertReference** claims object allows reference to a single X.509 certificate or a chain of X.509 certificates, either by providing the actual certificate data or by providing a hash reference for certificates that can be located in the target signature.

Name	Data Type	Value	Presence
`type`	**StringOrURI**	An identifier of the type of reference provided in the `ref` claim. The type identifier MUST be either one of the identifiers defined below, an identifier specified by the selected profile, or a URI identifier.	MANDATORY
`ref`	**[String]**	An array of string parameters according to conventions defined by the type identifier.	MANDATORY

The following type identifiers are defined:

Identifer	Ref Data Content
`chain`	Array of Base64 encoded X.509 certificates [RFC5280]. The certificates MUST be stored in the order starting with the end entity certificate. Any following certificate must be able to validate the signature on the previous certificate in the array.

Identifer	Ref Data Content
`chain_hash`	An array of one or two Base64 encoded hash values. The first hash value MUST be present and holds the hash over the signer's end entity X.509 certificate [RFC5280]. The second hash is the Base64 encoded hash over the complete certificate chain included in the target signature (including the signer's certificate). The chain hash is calculated over the concatenated bytes of the chain certificates exactly in the order they appear in the target signature. If the second hash value MAY be absent if the chain only contains the signer's certificate. The secound hash value MUST be present if the chain contains any certificates other than the signer's certificate.

> **Note**: All certificates referenced using the identifiers above are X.509 certificates. Profiles of this specification MAY define alternative types of public key containers. It should be noted however that a major function of these referenced certificates is not just to reference the public key, but also to provide the identity of the signer. It is therefore important for the full function of an SVT that the referenced public key container also provides the means to identify of the signer.

2.2.4. SVT JOSE Header

The SVT JWT MUST contain the following JOSE header parameters in accordance with section 5 of [RFC7519].

JOSE Header	Value
`typ`	This parameter MUST have the string value "JWT" (upper case).
`alg`	Specifying the algorithm used to sign the SVT JWT using a value specified in [RFC7518]. The specified signature hash algorithm MUST be identical to the hash algorithm specified in the **SigValAssertion** claims object `hash_algo` claim.

The SVT header MUST contain a public key or a reference to a public key used to verify the signature on the SVT in accordance with [RFC7515]. Each profile (See section 3.) MUST define the requirements for how the key or key reference is included in the header.

3. Profiles

Each signed document and signature type will have to define the precise content and use of several claims in the SVT.

Each profile MUST as a minimum define:

- How to specify reference to Signed Data content of the signed document.
- How to make reference to the target signature and the Signed Bytes of the signature.
- How references should be made to certificates supporting each siganture.
- How public keys or reference to public keys supporting validation of the signed SVT is included in the SVT.
- Whether each signature is supported by it's own SVT, or whether one SVT may support multiple signatures of the same document.
- Explicit information on how to perform signature validation based on an SVT, if applicable.
- How to attach an SVT to a document signature or signed document, if applicable.

4. Signature Validation with Signature Validation Token

Signature validation based on an SVT SHALL follow the following basic steps:

1. Locate all available tokens available for the signed document that is relevant for the target signature.
2. Select the most recent SVT that can be successfully validated and meets the requirement of the relying party.
3. Verify the integrity of the signature and the Signed Bytes of the target signature using the `sig_ref` claim.
4. Verify that the Signed Data reference in the original signature matches the reference values in the `sig_data_ref` claim.
5. Verify the integrity of referenced Signed Data using provided hash values in the `sig_data_ref` claim.
6. Obtain the verified certificates supporting the asserted signature validation through the `signer_cert_ref` claim.
7. Verify that signature validation policy results satisfy the requirements of the relying party.
8. Verify that verified time results satisfy the context within which the signed document is used.

After validating these steps, signature validity is established as well as the trusted signer certificate binding the identity of the signer to the signature.

5. Examples

The following example illustrates a basic SVT according to this specification issued for a signed PDF document.

Signature validation token JWT:

```
eyJraWQiOiJPZW5JKzQzNEpoYnZmRG50ZlZcLzhyT3hHN0ZrdnlqYUtWSmFWcUlGQlhvaFZoQWU1Zks4YW5vdjFTNjg4
cjdLYmFsK2Z2cGFIMWo4aWJnNTJRQnkxUFE9PSIsInR5cCI6IkpXVCIsImFsZyI6IlJTNTEyIn0.eyJhdWQiOiJodHRw
OlwvXC9leGFtcGxlLmNvbVwvYXVkaWVuY2UxIiwiaXNzIjoiaHR0cHM6XC9cL3N3ZWRlbmNvbm5lY3Quc2VcL3ZhbGlk
YXRvciIsImlhdCI6MTU4MjczMDY0NSwianRpIjoiZTIyYzViZTZkZDZjYzZkYjgzNGJjY2QwNjZmNWUyZTMiLCJzaWdf
dmFsX2NsYWltcyI6eyJzaWciOlt7ImV4dCI6bnVsbCwic2lnX3ZhbCI6W3sibXNnIjoiSW52YWxpZCBzaWduYXR1cmUi
LCJleHQiOm51bGwsInJlcyI6IkZBSUxFRCIsInBvbCI6Imh0dHA6XC9cL2lkLnN3ZWRlbmNvbm5lY3Quc2VcL3N2dFwv
c2lndmFsLXBvbGljeVwvY2hhaW5cLzAxIn1dLCJzaWdfcmVmIjp7InNpZ19oYXNoIjoiQmh1RTlCQ1lkcUxneW93bDJQ
Ym1uSzlkSkFtaVZ0VDF1OVZnaUY5OWgyaFZQekU0WExXdmJDUGU0YUNKM0l6RmZvTDlrM3RXcjBXK3d5OUJlcWlyY1E9
PSIsImlkIjpudWxsLCJzYl9oYXNoIjoiYnVlcTVIVE8xYnRwQ3JYUlg3VHpFS1VyTkpRaEdHOHFCaDR3eEVTcVJMM0J6
bjRjbHZLMzdqWXUwS2pNTWtnSlFFTWZBMWIzaW1peTc5dDdoK1loOHc9PSJ9LCJzaWduZXJfY2VydF9yZWYiOnsicmVm
IjpbIk5TdUZNXC92SitiZUJsUXRRVHptY1loNXg3TDhXQzlFMUtQSFJBMWlvTk9sS1ZHYmxhOVVSelljc2lzQXgyYmNz
cU9oa3ZWVGMzbUs5RTZhZzA3aGZhdz09Il0sInR5cGUiOiJjaGFpbl9oYXNoIn0sInNpZ19kYXRhX3JlZiI6W3sicmVm
IjoiMCAxMjI5MzUgMTI3OTM3IDI3NDMwIiwiaGFzaCI6Imt1VWI4NkZzTU5tSmwzdjRiUUswOUZrUWd2bzlReDAxbk5S
eVFLVVppaEdFdW1kVnF0dUJLTlBxWkkxVHpDUWV3Nm44b0ZNak5oQjhDMFhNSmxrRE9RPT0ifV0sInRpbWVfdmFsIjpb
XX1dLCJleHQiOnsibmFtZTIiOiJ2YWwyIiwibmFtZTEiOiJ2YWwxIn0sInZlciI6IjEuMCIsInByb2ZpbGUiOiJQREYi
LCJoYXNoX2FsZ28iOiJodHRwOlwvXC93d3cudzMub3JnXC8yMDAxXC8wNFwveG1sZW5jI3NoYTUxMiJ9fQ.DhrCRxT_U
8LeqK1BU9-5Bqui2cs5n21PrSqPnDtVa7mxUtqTnouOXjVfuWR0lfNAjEkc1y2QSX5x2dmMdCpNLWX127UHYiAm8NzeY
uoWqdnxKiy61hZ1l0Jldnk52ngG_2UNDnrCGBo9OgC90kG2bFQimZB3WgVtE7ad_HAwIXwd-vEHt6Sf2yWXlUTYqQ1Dx
gq6pTKQnuf5ahsHVyeDihgNeix8-cGx1MEvvHNUpCcIXBx67BEcZ-SrqRoIZkVqEcW83KFMgqKWmWDgp4z_CKM5ix2dV
zwp1GvYOM6M3QUKYgmiNA6dMWJvXeJZ-KKi5A-6gEqfgOsixuZechcDon_3nMzEeNBSJFXU7ohkvxIJN9LXNFAxzAT2U
mASxrL9wvaQMmJHYMeet-vUsOPWcsq07eKO5bnsYwrs9igYeotgcT_Nl74Rmf9uMye_IgyzlS_NLL4xq9Aaf6LPXWZ0S
_plugvfzv7HuzXNOY994voq8sOpO9xKYhqYnzbdDFKU
```

Decoded JWT Header:

```
{
  "kid":"OenI+434JhbvfDntfV\/8rOxG7FkvyjaKVJaVqIFBXohVhAe5fK8anov1S688r7Kbal+fvpaH1j8ibg52QB
         y1PQ==",
  "typ" : "JWT",
  "alg" : "RS512"
}
```

Decoded JWT Claims:

```
{
  "aud" : "http://example.com/audience1",
  "iss" : "https://swedenconnect.se/validator",
  "iat" : 1582730645,
  "jti" : "e22c5be6dd6cc6db834bccd066f5e2e3",
  "sig_val_claims" : {
    "sig" : [ {
      "ext" : null,
      "sig_val" : [ {
        "msg" : "Invalid signature",
        "ext" : null,
        "res" : "FAILED",
        "pol" : "http://id.swedenconnect.se/svt/sigval-policy/chain/01"
```

```
      } ],
      "sig_ref" : {
        "sig_hash" : "BhuE9BCYdqLgyowl2PbmnK9dJAmiVtT1u9VgiF99h2hVPzE4XLWvbCPe4aCJ3IzFfoL9k3
                      tWr0W+wy9BeqircQ==",
        "id" : null,
        "sb_hash" : "bueq5HTO1btpCrXRX7TzEKUrNJQhGG8qBh4wxESqRL3Bzn4clvK37jYu0KjMMkgJQEMfA1b
                     3imiy79t7h+Yh8w=="
      },
      "signer_cert_ref" : {
        "ref" : [ "NSuFM/vJ+beBlQtQTzmcYh5x7L8WC9E1KPHRA1ioNOlKVGbla9URzYcsisAx2bcsqOhkvVTc3
                   mK9E6ag07hfaw==" ],
        "type" : "chain_hash"
      },
      "sig_data_ref" : [ {
        "ref" : "0 122935 127937 27430",
        "hash" : "kuUb86FsMNmJl3v4bQK09FkQgvo9Qx01nNRyQKUZihGEumdVqtuBKNPqZI1TzCQew6n8oFMjNh
                  B8C0XMJlkDOQ=="
      } ],
      "time_val" : [ ]
    } ],
    "ext" : {
      "name2" : "val2",
      "name1" : "val1"
    },
    "ver" : "1.0",
    "profile" : "PDF",
    "hash_algo" : "http://www.w3.org/2001/04/xmlenc#sha512"
  }
}
```


> Note: Line breaks in the decoded example are inserted for readablilty. These are not allowed in valid JSON data.

6. Normative References

[RFC2119]

> Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, March 1997.

[RFC5280]

> D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008.

[RFC6931]

> Eastlake 3rd, D., Additional XML Security Uniform Resource Identifiers (URIs), April 2013.

[RFC7515]

> Jones, M., Bradley, J., Sakimura, N., JSON Web Signature (JWS), May 2015.

[RFC7518]

> Jones, M., JSON Web Algorithms (JWA), May 2015.

[RFC7519]

> Jones, M., Bradley, J., Sakimura, N., JSON Web Token (JWT), May 2015.

[RFC8174]

> Leiba, B., Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words, May 2017.

[ETSI EN 319 102-1]

> ETSI - Electronic Signatures and Infrastructures (ESI); Procedures for Creation and Validation of AdES Digital Signatures; Part 1: Creation and Validation.