Stream: Independent Submission
RFC: 9321
Category: Informational
Published: October 2022
ISSN: 2070-1721
Authors: S. Santesson, *IDsec Solutions*; R. Housley, *Vigil Security*

RFC 9321
Signature Validation Token

Abstract

Electronic signatures have a limited lifespan with respect to the time period that they can be validated and determined to be authentic. The Signature Validation Token (SVT) defined in this specification provides evidence that asserts the validity of an electronic signature. The SVT is provided by a trusted authority, which asserts that a particular signature was successfully validated according to defined procedures at a certain time. Any future validation of that electronic signature can be satisfied by validating the SVT without any need to also validate the original electronic signature or the associated digital certificates. The SVT supports electronic signatures in Cryptographic Message Syntax (CMS), XML, PDF, and JSON documents.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9321.

Copyright Notice

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 certificate containing the signer's public key is not available.
- Trusted information about the time when the signature was actually created is not available.
- Algorithms used to create the electronic signature may no longer be considered secure at the time of validation and may therefore no longer be available in software libraries.
- Services necessary to validate the signature are no longer available at the time of validation.
- Supporting evidence such as certification authority (CA) certificates, Online Certificate Status Protocol (OCSP) responses, Certificate Revocation Lists (CRLs), or timestamps is not available or can't be validated.

The challenges to validation of an electronic signature increase over time, and eventually it may simply be impossible to verify the signature with a sufficient level of assurance.

Existing standards, such as the ETSI XAdES [XADES] profile for XML signatures [XMLDSIG11], ETSI PAdES [PADES] profile for PDF signatures [ISOPDF2], and ETSI CAdES [CADES] profile for CMS signatures [RFC5652], can be used to extend the time within which a signature can be validated at the cost of significant complexity, which involves storing and validating significant amounts of external evidence data such as revocation data, signature time stamps, and archival time stamps.

The Signature Validation Token (SVT) defined in this specification takes a trusted signature validation process as an input and preserves the validation result for the associated signature and signed document. The SVT asserts that a particular electronic signature was successfully validated by a trusted authority according to defined procedures at a certain time. Those procedures **MUST** include checks that the signature match the signed document, checks that the signature can be validated by the signing certificate, and checks that the signing certificate pass certificate path validation [RFC5280]. Those procedures **MAY** also include checks associated with a particular trust policy such as that an acceptable certificate policy [RFC5280] [RFC3647] was used to issue the signer's certificate and checks that an acceptable signature policy was used by the signer [RFC3125].

Once the SVT is issued by a trusted authority, any future validation of that electronic signature can be satisfied by validating the SVT without any need to also revalidate the original electronic signature.

As the SVT is used to preserve validation results obtained through applying existing standards for signature validation, it is complementary to and not a replacement for such standards, including the ETSI standards for long-term validation listed above. The SVT does, however, have the potentially positive effect that it may significantly reduce the need to apply complex long-term validation and preservation techniques for signature validation if an SVT is issued and applied to the signed document at an early stage where the signature can be validated without support of large amounts of external evidence. The use of SVTs may therefore drastically reduce the complexity of revalidation of old archived electronic signatures.

The SVT can be signed with private keys and algorithms that provide confidence for a considerable time period. In fact, multiple SVTs can be used to offer greater assurance. For example, one SVT could be produced with a large RSA private key, a second one with a strong elliptic curve, and a third one with a quantum safe digital signature algorithm to protect against advances in computing power and cryptanalytic capabilities. Further, the trusted authority can add additional SVTs in the future using fresh private keys and signatures to extend the lifetime of the SVTs if necessary.

2. Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**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.

This document use the following terms:

Signed Data: The data covered by a particular electronic signature. This is typically equivalent to the signed content of a document, and it 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. In JSON Web Signature (JWS), this is the unencoded payload data (before base64url encoding).

Signed Bytes: These are the actual bytes of data that were hashed and signed by the digital 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. In CMS and PDF signatures, this is the DER-encoded SignedAttributes structure. In JWS, this is the protected header and payload data formatted according to [RFC7515].

When these terms are used as defined in this section, they appear with a capitalized first letter.

3. Signature Validation Token

3.1. Signature Validation Token Function

The Signature Validation Token (SVT) is created by a trusted service to assert evidence of successful electronic signature validation using a well-defined and trustworthy signature validation process. The SVT binds the validation result to the validated signature, the document signed by the signature, and the certificate of the signer. This allows a relying party to verify the validity of a signed document without having to revalidate the original signature or to reuse any of its associated cryptographic algorithms for as long as the SVT itself can be validated. The SVT achieves this by binding the following information to a specific electronic signature:

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

The SVT aims to support long-term validation that can be further extended into the future by applying the following strategies:

- by using secure algorithms with long life expectancy when signing the SVT
- by reissuing the SVT before it becomes insecure or is considered expired

- optionally, by issuing multiple SVTs with different algorithms to provide redundancy in case one algorithm is broken

3.2. Signature Validation Token Syntax

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

3.2.1. Data Types

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

String: JSON Data Type of string that contains an arbitrary case-sensitive string value.

Base64Binary: JSON Data Type of string that contains a Base64-encoded byte array of binary data.

StringOrURI: JSON Data Type of string that contains an arbitrary string or a URI as defined in [RFC7519]. It is **REQUIRED** to contain the colon character (":") to be a URI.

URI: JSON Data Type of string that contains a URI as defined in [RFC7519].

Integer: JSON Data Type of number that contains a 32-bit signed integer value (from -2^{31} to $2^{31}-1$).

Long: JSON Data Type of number that contains a 64-bit signed integer value (from -2^{63} to $2^{63}-1$).

NumericDate: JSON Data Type of number that contains data as defined in [RFC7519], which is the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds.

Boolean: JSON Data Type of boolean that contains the explicit value of true or false.

Object<Class>: A JSON object holding a claims object of a class defined in this specification (see Section 3.2.2).

Map<Type>: A JSON object with name-value pairs where the value is an object of the specified Type in the notation. For example, Map<String> is a JSON object with name-value pairs where all values are of type String.

Array: A JSON array of a specific data type as defined in this section. 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: A JSON null that represents an absent value. A claim with a null value is equivalent with an absent claim.

3.2.2. Signature Validation Token JWT Claims

The SVT **MUST** contain only JWT claims in the following list:

"jti": A String data type that is 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. An SVT **MUST** contain one "JWT ID" claim.

"iss": A StringOrURI data type that is an "Issuer" registered claim according to [RFC7519], which is an arbitrary unique identifier of the SVT issuer. This value **SHOULD** have the value of a URI based on a domain owned by the issuer. An SVT **MUST** contain one "Issuer" claim.

"iat": A NumericDate data type that is an "Issued At" registered claim according to [RFC7519], which expresses the time when this SVT was issued. An SVT **MUST** contain one "Issued At" claim.

"aud": A [StringOrURI] data type or a StringOrURI data type that is 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 data type value instead of as an array of values. Inclusion of the "Audience" claim in an SVT is **OPTIONAL**.

"exp": A NumericDate data type that is an "Expiration Time" registered claim according to [RFC7519], which expresses the time when services and responsibilities related to this SVT are no longer provided by the SVT issuer. The precise meaning of the expiration time claim is defined by local policies. See implementation note below. Inclusion of the "Expiration Time" claim in an SVT is **OPTIONAL**.

"sig_val_claims": An Object<SigValidation> data type that contains signature validation claims for this SVT extending the standard registered JWT claims above. An SVT **MUST** contain one sig_val_claims claim.

Note: An SVT asserts that a particular validation process was undertaken at a stated time. This fact never changes and never expires. However, some other 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.

3.2.3. SigValidation Object Class

The sig_val_claims JWT claim uses the SigValidation object class. A SigValidation object holds all custom claims, and a SigValidation object contains the following parameters:

"ver": A String data type representing the version. This parameter **MUST** be present and the version in this specification indicated by the value "1.0".

"profile": A StringOrURI data type representing the name of a profile that defines conventions followed for specific claims and any extension points used by the SVT issuer. This parameter **MUST** be present.

"hash_algo": A URI data type that identifies the hash algorithm used to compute the hash values within the SVT. The URI identifier **MUST** be one defined in [RFC9231] or in the IANA registry defined by this specification. This parameter **MUST** be present.

"sig": An [Object<Signature>] data type that gives information about validated electronic 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.

"ext": A Map<String> data type that provides additional claims related to the SVT. Extension claims are added at the discretion of the SVT issuer; however, extension claims **MUST** follow any conventions defined in a profile of this specification (see Section 4). Inclusion of this parameter is **OPTIONAL**.

3.2.4. Signature Claims Object Class

The sig parameter in the SigValidation object class uses the Signature object class. The Signature object contains claims related to signature validation evidence for one signature, and it contains the following parameters:

"sig_ref": An Object<SigReference> data type that contains reference information identifying the target signature. This parameter **MUST** be present.

"sig_data_ref": An [Object<SignedDataReference>] data type that contains an array of references to Signed Data that was signed by the target electronic signature. At least one SignedDataReference object **MUST** be present.

"signer_cert_ref": An Object<CertReference> data type that references the signer's certificate and optionally references a supporting certification path that was used to verify the target electronic signature. This parameter **MUST** be present.

"sig_val": An [Object<PolicyValidation>] data type that contains an array of results of signature verification according to defined procedures. At least one PolicyValidation object **MUST** be present.

"time_val": An [Object<TimeValidation>] data type that contains an array of time verification results showing that the target signature has existed at a specific time in the past. Inclusion of this parameter is **OPTIONAL**.

"ext": A MAP<String> data type that provides additional claims related to the target signature. Extension claims are added at the discretion of the SVT issuer; however, extension claims **MUST** follow any conventions defined in a profile of this specification (see Section 4). Inclusion of this parameter is **OPTIONAL**.

3.2.5. SigReference Claims Object Class

The sig_ref parameter in the Signature object class uses the SigReference object class. The SigReference object provides information used to match the Signature claims object to a specific target electronic signature and to verify the integrity of the target signature value and Signed Bytes, and it contains the following parameters:

"id": A String data type that contains an identifier assigned to the target signature. Inclusion of this parameter is **OPTIONAL**.

"sig_hash": A Base64Binary data type that contains a hash value of the target electronic signature value. This parameter **MUST** be present.

"sb_hash": A Base64Binary data type that contains a hash value of the Signed Bytes of the target electronic signature. This parameter **MUST** be present.

3.2.6. SignedDataReference Claims Object Class

The sig_data_ref parameter in the Signature object class uses the SignedDataReference object class. The SignedDataReference object provides information used to verify the target electronic signature references to Signed Data as well as to verify the integrity of all data that is signed by the target signature, and it contains the following parameters:

"ref": A String data type that contains a reference identifier for the data or data fragment covered by the target electronic signature. This parameter **MUST** be present.

"hash": A Base64Binary data type that contains the hash value for the data covered by the target electronic signature. This parameter **MUST** be present.

3.2.7. PolicyValidation Claims Object Class

The sig_val parameter in the Signature object class uses the PolicyValidation object class. The PolicyValidation object provides information about the result of a validation process according to a specific policy, and it contains the following parameters:

"pol": A StringOrURI data type that contains the identifier of the policy governing the electronic signature verification process. This parameter **MUST** be present.

"res": A String data type that contains the result of the electronic signature verification process. The value **MUST** be one of "PASSED", "FAILED", or "INDETERMINATE" as defined by [ETSI319102-1]. This parameter **MUST** be present.

"msg": A String data type that contains a message describing the result. Inclusion of this parameter is **OPTIONAL**.

"ext": A MAP<String> data type that provides additional claims related to the target signature. Extension claims are added at the discretion of the SVT issuer; however, extension claims **MUST** follow any conventions defined in a profile of this specification (see Section 4). Inclusion of this parameter is **OPTIONAL**.

3.2.8. TimeValidation Claims Object Class

The time_val parameter in the Signature object class uses the TimeValidation object class. The TimeValidation claims object provides information about the result of validating evidence of time asserting that the target signature existed at a particular time in the past. Evidence of time

is typically a timestamp according to [RFC3161], but other types of evidence may be used such as a previously issued SVT for this signature. The TimeValidation claims object contains the following parameters:

"time": A NumericDate data type that contains the verified time. This parameter **MUST** be present.

"type": A StringOrURI data type that contains an identifier of the type of evidence of time. This parameter **MUST** be present.

"iss": A StringOrURI data type that contains an identifier of the entity that issued the evidence of time. This parameter **MUST** be present.

"id": A String data type that contains an unique identifier assigned to the evidence of time. Inclusion of this parameter is **OPTIONAL**.

"hash": A Base64Binary data type that contains the hash value of the validated evidence of time. Inclusion of this parameter is **OPTIONAL**.

"val": An [Object<PolicyValidation>] data type that contains an array of results of the time evidence validation according to defined validation procedures. Inclusion of this parameter is **OPTIONAL**.

"ext": A MAP<String> data type that provides additional claims related to the target signature. Extension claims are added at the discretion of the SVT issuer; however, extension claims **MUST** follow any conventions defined in a profile of this specification (see Section 4). Inclusion of this parameter is **OPTIONAL**.

3.2.9. CertReference Claims Object Class

The signer_cert_ref parameter in the Signature object class uses the CertReference object class. The CertReference object references a single X.509 certificate or a X.509 certification path either by providing the certificate data or by providing hash references for certificates that can be located in the target electronic signature, and it contains the following parameters:

"type": A StringOrURI data type that contains an identifier of the type of reference. The type identifier **MUST** be one of the identifiers defined below, an identifier specified by the selected profile, or a URI identifier. This parameter **MUST** be present.

"ref": A [String] data type that contains an array of string parameters according to conventions defined by the type identifier. At least one parameter **MUST** be present.

The following type identifiers are defined:

"chain": The ref contains an array of Base64-encoded X.509 certificates [RFC5280]. The certificates **MUST** be provided 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.

"chain_hash": The ref contains an array of one or more Base64-encoded hash values where each hash value is a hash over a X.509 certificate [RFC5280] used to validate the signature. The certificates **MUST** be provided 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. This option **MUST NOT** be used unless all hashed certificates are present in the target electronic signature.

Note: All certificates referenced using the identifiers above are X.509 certificates. Profiles of this specification **MAY** define alternative types of public key containers; however, a major function of these referenced certificates is not just to reference the public key but also to provide the subject name 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 the signer.

3.2.10. SVT JOSE Header

The SVT JWT **MUST** contain the following JSON Object Signing and Encryption (JOSE) header parameters in accordance with Section 5 of [RFC7519]:

"typ": This parameter **MUST** have the string value "JWT" (upper case).

"alg": This parameter identifies the algorithm used to sign the SVT JWT. The algorithm identifier **MUST** be specified in [RFC7518] or the IANA "JSON Web Signature and Encryption Algorithms" registry [IANA-JOSE-REG]. The specified signature hash algorithm **MUST** be identical to the hash algorithm specified in the hash_algo parameter of the SigValidation object within the sig_val_claims 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, as discussed in Section 4, **MUST** define the requirements for how the key or key reference is included in the header.

4. Profiles

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

At a minimum, each profile **MUST** define:

- The identifier of the profile
- How to reference the Signed Data content of the signed document
- How to reference the target electronic signature and the Signed Bytes of the signature
- How to reference certificates supporting each electronic signature
- How to include public keys or references to public keys in the SVT
- Whether each electronic signature is supported by a single SVT, or one SVT may support multiple electronic signatures of the same document

A profile **MAY** also define:

- Explicit information on how to perform signature validation based on an SVT
- How to attach an SVT to an electronic signature or signed document

4.1. Defined Profiles

The following profiles are defined in appendixes of this document:

Appendix A: XML Signature Profile

Appendix B: PDF Signature Profile

Appendix C: JWS Profile

Other documents **MAY** define other profiles that **MAY** complement, amend, or supersede these profiles.

5. Signature Verification with an SVT

Signature verification based on an SVT **MUST** follow these steps:

1. Locate all available SVTs available for the signed document that are relevant for the target electronic 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 electronic signature using the sig_ref claim.
4. Verify that the Signed Data reference in the original electronic 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 electronic signature verification 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 for the use of the signed document.

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

6. IANA Considerations

6.1. Claim Names Registration

IANA has registered the "sig_val_claims" claim name in the "JSON Web Token Claims" registry established by Section 10.1 of [RFC7519].

6.1.1. Registry Contents

Claim Name: sig_val_claims

Claim Description: Signature Validation Token

Change Controller: IESG

Specification Document(s): Section 3.2.3 of RFC 9321

6.2. Header Parameter Names Registration

IANA has registered the "svt" Header Parameter in the "JSON Web Signature and Encryption Header Parameters" registry established by [RFC7515].

6.2.1. Registry Contents

Header Parameter Name: svt

Header Parameter Description: Signature Validation Token

Header Parameter Usage Location(s): JWS

Change Controller: IESG

Specification Document(s): Appendix C.1.1 of RFC 9321

7. Security Considerations

7.1. Level of Reliance

An SVT allows a signature verifier to still validate the original signature using the original signature data and to use the information in the SVT selectively to confirm the validity and integrity of the original data, such as confirming the integrity of Signed Data or the validity of the signer's certificate, etc.

Another way to use an SVT is to completely rely on the validation conclusion provided by the SVT and to omit revalidation of the original signature value and original certificate status checking data.

This choice is a decision made by the verifier according to its own policy and risk assessment.

However, even when relying on the SVT validation conclusion of an SVT, it is vital to still verify that the present SVT is correctly associated with the document and signature that is being validated by validating the hashed reference data in the SVT of the signature, signing certificate chain, Signed Data, and the Signed Bytes.

7.2. Aging Algorithms

Even if the SVT provides protection against algorithms becoming weakened or broken over time, this protection is only valid for as long as the algorithms used to sign the SVT are still considered secure. It is advisable to reissue SVTs in cases where an algorithm protecting the SVT is getting close to its end of life.

One way to increase the resistance of algorithms becoming insecure, is to issue multiple SVTs for the same signature with different algorithms and key lengths where one algorithm could still be secure even if the corresponding algorithm used in the alternative SVT is broken.

8. References

8.1. Normative References

[CADES] ETSI, "Electronic Signatures and Infrastructures (ESI); CAdES digital signatures; Part 1: Building blocks and CAdES baseline signatures", v1.1.1, ETSI EN 319 122-1, April 2016.

[ETSI319102-1] ETSI, "Electronic Signatures and Infrastructures (ESI); Procedures for Creation and Validation of AdES Digital Signatures; Part 1: Creation and Validation", v1.1.1, ETSI EN 319 102-1, May 2016.

[IANA-JOSE-REG] IANA, "JSON Object Signing and Encryption (JOSE)", <https://www.iana.org/assignments/jose/>.

[ISOPDF2] ISO, "Document management -- Portable document format -- Part 2: PDF 2.0", ISO 32000-2:2020, December 2020.

[PADES] ETSI, "Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 1: Building blocks and PAdES baseline signatures", v1.1.1, ETSI EN 319 142-1, April 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3125] Ross, J., Pinkas, D., and N. Pope, "Electronic Signature Policies", RFC 3125, DOI 10.17487/RFC3125, September 2001, <https://www.rfc-editor.org/info/rfc3125>.

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <https://www.rfc-editor.org/info/rfc3161>.

[RFC3647] Chokhani, S., Ford, W., Sabett, R., Merrill, C., and S. Wu, "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework", RFC 3647, DOI 10.17487/RFC3647, November 2003, <https://www.rfc-editor.org/info/rfc3647>.

[RFC5035] Schaad, J., "Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility", RFC 5035, DOI 10.17487/RFC5035, August 2007, <https://www.rfc-editor.org/info/rfc5035>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9231] Eastlake 3rd, D., "Additional XML Security Uniform Resource Identifiers (URIs)", RFC 9231, DOI 10.17487/RFC9231, July 2022, <https://www.rfc-editor.org/info/rfc9231>.

[XADES] ETSI, "Electronic Signatures and Infrastructures (ESI); XAdES digital signatures; Part 1: Building blocks and XAdES baseline signatures", v1.1.1, ETSI EN 319 132-1, April 2016.

[XMLDSIG11] Eastlake 3rd, D., Reagle, J., Solo, D., Hirsch, F., Nystrom, M., Roessler, T., and K. Yiu, "XML Signature Syntax and Processing Version 1.1", W3C Proposed Recommendation, April 2013. Latest version available at https://www.w3.org/TR/xmldsig-core1/.

8.2. Informative References

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

Appendix A. XML Signature Profile

This appendix defines a profile for implementing SVTs with a signed XML document and defines the following aspects of SVT usage:

- How to include reference data related to XML signatures and XML documents in an SVT
- How to add an SVT token to an XML signature

XML documents can have any number of signature elements, signing an arbitrary number of fragments of XML documents. The actual signature element may be included in the signed XML document (enveloped), include the Signed Data (enveloping), or may be separate from the signed content (detached).

To provide a generic solution for any type of XML signature, an SVT is added to each XML signature element within the XML signature <ds:Object> element.

A.1. Notation

A.1.1. References to XML Elements from XML Schemas

When referring to elements from the W3C XML Signature namespace (https://www.w3.org/2000/09/xmldsig#), the following syntax is used:

- <ds:Signature>

When referring to elements from the ETSI XAdES XML Signature namespace (https://uri.etsi.org/01903/v1.3.2#), the following syntax is used:

- <xades:CertDigest>

When referring to elements defined in this specification (http://id.swedenconnect.se/svt/1.0/sig-prop/ns), the following syntax is used:

- <svt:Element>

A.2. SVT in XML Documents

When SVTs are provided for XML signatures, then one SVT **MUST** be provided for each XML signature.

An SVT embedded within the XML signature element **MUST** be placed in a <svt:SignatureValidationToken> element as defined in Appendix A.2.1.

A.2.1. SignatureValidationToken Signature Property

The <svt:SignatureValidationToken> element **MUST** be placed in a <ds:SignatureProperty> element in accordance with [XMLDSIG11]. The <ds:SignatureProperty> element **MUST** be placed inside a <ds:SignatureProperties> element inside a <ds:Object> element inside a <ds:Signature> element.

Note: [XMLDSIG11] requires the Target attribute to be present in <ds:SignatureProperty>, referencing the signature targeted by this signature property. If an SVT is added to a signature that does not have an Id attribute, implementations **SHOULD** add an Id attribute to the <ds:Signature> element and reference that Id in the Target attribute. This Id attribute and Target attribute value matching is required by the [XMLDSIG11] standard, but it is redundant in the context of SVT validation as the SVT already contains information that uniquely identifies the target signature. Validation applications **SHOULD NOT** reject an SVT token because of Id and Target attribute mismatch and **MUST** rely on matching against a signature using signed information in the SVT itself.

The <svt:SignatureValidationToken> element is defined by the following XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://id.swedenconnect.se/svt/1.0/sig-prop/ns"
    xmlns:svt="http://id.swedenconnect.se/svt/1.0/sig-prop/ns">

  <xs:element name="SignatureValidationToken"
      type="svt:SignatureValidationTokenType" />

  <xs:complexType name="SignatureValidationTokenType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

</xs:schema>
```

The SVT token **MUST** be included as a string representation of the SVT JWT. Note that this is the string representation of the JWT without further encoding. The SVT **MUST NOT** be represented by the Base64-encoded bytes of the JWT string.

Example:

```
<ds:Signature Id="MySignatureId">
  ...
  <ds:Object>
    <ds:SignatureProperties>
      <ds:SignatureProperty Target="#MySignatureId">
        <svt:SignatureValidationToken>
              eyJ0eXAiOiJKV1QiLCJhb...2aNZ
        </svt:SignatureValidationToken>
      </ds:SignatureProperty>
    </ds:SignatureProperties>
  </ds:Object>
</ds:Signature>
```

A.2.2. Multiple SVTs in an XML Signature

If a new SVT is stored in a signature that already contains a previously issued SVT, implementations can choose either to replace the existing SVT or to store the new SVT in addition to the existing SVT.

If the new SVT is stored in addition to the old SVT, it **SHOULD** be stored in a new <ds:SignatureProperty> element inside the existing <ds:SignatureProperties> element where the old SVT is located.

For interoperability robustness, signature validation applications **MUST** be able to handle signatures where the new SVT is located in a new <ds:Object> element.

A.3. XML Signature SVT Claims

A.3.1. XML Profile Identifier

When this profile is used, the SigValidation object **MUST** contain a "profile" claim with the value "XML".

A.3.2. XML Signature Reference Data

The SVT Signature object **MUST** contain a "sig_ref" claim (SigReference object) with the following elements:

"id": The Id-attribute of the XML signature, if present.

"sig_hash": The hash over the signature value bytes.

"sb_hash": The hash over the canonicalized <ds:SignedInfo> element (the bytes the XML signature algorithm has signed to generate the signature value).

A.3.3. XML Signed Data Reference Data

The SVT Signature object **MUST** contain one instance of the "sig_data" claim (SignedData object) for each <ds:Reference> element in the <ds:SignedInfo> element. The "sig_data" claim **MUST** contain the following elements:

"ref": The value of the URI attribute of the corresponding <ds:Reference> element.

"hash": The hash of all bytes that were identified by the corresponding <ds:Reference> element after applying all identified canonicalization and transformation algorithms. These are the same bytes that are hashed by the hash value in the <ds:DigestValue> element inside the <ds:Reference> element.

A.3.4. XML Signer Certificate References

The SVT Signature object **MUST** contain a "signer_cert_ref" claim (CertReference object). The "type" parameter of the "signer_cert_ref" claim **MUST** be either "chain" or "chain_hash".

- The "chain" type **MUST** be used when signature validation was performed using one or more certificates where some or all of the certificates in the chain are not present in the target signature.
- The "chain_hash" type **MUST** be used when signature validation was performed using one or more certificates where all of the certificates are present in the target signature.

A.4. JOSE Header

A.4.1. SVT Signing Key Reference

The SVT JOSE header for XML signatures must contain one of the following header parameters in accordance with [RFC7515] for storing a reference to the public key used to verify the signature on the SVT:

"x5c": Holds an X.509 certificate [RFC5280] or a chain of certificates. The certificate holding the public key that verifies the signature on the SVT **MUST** be the first certificate in the chain.

"kid": A key identifier holding the Base64-encoded hash value of the certificate that can verify the signature on the SVT. The hash algorithm **MUST** be the same hash algorithm used when signing the SVT as specified by the "`alg`" Header Parameter.

Appendix B. PDF Signature Profile

This appendix defines a profile for implementing SVTs with a signed PDF document, and it defines the following aspects of SVT usage:

- How to include reference data related to PDF signatures and PDF documents in an SVT.
- How to add an SVT token to a PDF document.

PDF document signatures are added as incremental updates to the signed PDF document and signs all data of the PDF document up until the current signature. When more than one signature is added to a PDF document the previous signature is signed by the next signature and can not be updated with additional data after this event.

To minimize the impact on PDF documents with multiple signatures and to stay backwards compatible with PDF software that does not understand SVTs, PDF documents add one SVT token for all signatures of the PDF as an extension to a document timestamp added to the signed PDF as an incremental update. This SVT covers all signatures of the signed PDF.

B.1. SVTs in PDF Documents

The SVT for a signed PDF document **MAY** provide signature validation information about any of the present signatures in the PDF. The SVT **MUST** contain a separate "sig" claim (Signature object) for each signature on the PDF that is covered by the SVT.

An SVT added to a signed PDF document **MUST** be added to a document timestamp in accordance with ISO 32000-2:2020 [ISOPDF2].

The document timestamp contains an [RFC3161] timestamp token (TSTInfo) in EncapsulatedContentInfo of the CMS signature. The SVT **MUST** be added to the timestamp token (TSTInfo) as an Extension object as defined in Appendix B.1.1.

B.1.1. SVT Extension to Timestamp Tokens

The SVT extension is an Extension suitable to be included in TSTInfo as defined by [RFC3161].

The SVT extension is identified by the Object Identifier (OID) 1.2.752.201.5.2.

This extension data (OCTET STRING) holds the bytes of SVT JWT, represented as a UTF-8-encoded string.

This extension **MUST NOT** be marked critical.

Note: Extensions in timestamp tokens according to [RFC3161] are imported from the definition of the X.509 certificate extensions defined in [RFC5280].

B.2. PDF Signature SVT Claims

B.2.1. PDF Profile Identifier

When this profile is used, the SigValidation object **MUST** contain a "profile" claim with the value "PDF".

B.2.2. PDF Signature Reference Data

The SVT Signature object **MUST** contain a "sig_ref" claim (SigReference object) with the following elements:

"id": Absent or a Null value.

"sig_hash": The hash over the signature value bytes.

"sb_hash": The hash over the DER-encoded SignedAttributes in SignerInfo.

B.2.3. PDF Signed Data Reference Data

The SVT Signature object **MUST** contain one instance of the "sig_data" claim (SignedData object) with the following elements:

"ref": The string representation of the ByteRange value of the PDF signature dictionary of the target signature. This is a sequence of integers separated by space where each integer pair specifies the start index and length of a byte range.

"hash": The hash of all bytes identified by the ByteRange value. This is the concatenation of all byte ranges identified by the ByteRange value.

B.2.4. PDF Signer Certificate References

The SVT Signature object **MUST** contain a "signer_cert_ref" claim (CertReference object). The "type" parameter of the "signer_cert_ref" claim **MUST** be either "chain" or "chain_hash".

- The "chain" type **MUST** be used when signature validation was performed using one or more certificates where some or all of the certificates in the chain are not present in the target signature.
- The "chain_hash" type **MUST** be used when signature validation was performed using one or more certificates where all of the certificates are present in the target signature.

Note: The referenced signer certificate **MUST** match any certificates referenced using ESSCertID or ESSCertIDv2 from [RFC5035].

B.3. JOSE Header

B.3.1. SVT Signing Key Reference

The SVT JOSE header must contain one of the following header parameters in accordance with [RFC7515] for storing a reference to the public key used to verify the signature on the SVT:

"x5c": Holds an X.509 certificate [RFC5280] or a chain of certificates. The certificate holding the public key that verifies the signature on the SVT **MUST** be the first certificate in the chain.

"kid": A key identifier holding the Base64-encoded hash value of the certificate that can verify the signature on the SVT. The hash algorithm **MUST** be the same hash algorithm used when signing the SVT as specified by the "`alg`" Header Parameter. The referenced certificate **SHOULD** be the same certificate that was used to sign the document timestamp that contains the SVT.

Appendix C. JWS Profile

This appendix defines a profile for implementing SVTs with a JWS signed payload according to [RFC7515], and it defines the following aspects of SVT usage:

- How to include reference data related to JWS signatures in an SVT.
- How to add an SVT token to JWS signatures.

A JWS may have one or more signatures, depending on its serialization format, signing the same payload data. A JWS either contains the data to be signed (enveloping) or may sign any externally associated payload data (detached).

To provide a generic solution for JWS, an SVT is added to each present signature as a JWS Unprotected Header. If a JWS includes multiple signatures, then each signature includes its own SVT.

C.1. SVT in JWS

An SVT token **MAY** be added to any signature of a JWS to support validation of that signature. If more than one signature is present, then each present SVT **MUST** provide information exclusively related to one associated signature and **MUST NOT** include information about any other signature in the JWS.

Each SVT is stored in its associated signature's "svt" header as defined in Appendix C.1.1.

C.1.1. "svt" Header Parameter

The "svt" (Signature Validation Token) Header Parameter is used to contain an array of SVT tokens to support validation of the associated signature. Each SVT token in the array has the format of a JWT as defined in [RFC7519] and is stored using its natural string representation without further wrapping or encoding.

The "svt" Header Parameter, when used, **MUST** be included as a JWS Unprotected Header.

Note: A JWS Unprotected Header is not supported with JWS Compact Serialization. A consequence of adding an SVT token to a JWS is therefore that JWS JSON Serialization **MUST** be used either in the form of general JWS JSON Serialization (for one or more signatures) or in the form of flattened JWS JSON Serialization (optionally used when only one signature is present in the JWS).

C.1.2. Multiple SVTs in a JWS Signature

If a new SVT is stored in a signature that already contains a previously issued SVT, implementations can choose either to replace the existing SVT or to store the new SVT in addition to the existing SVT.

If a JWS signature already contains an array of SVTs and a new SVT is to be added, then the new SVT **MUST** be added to the array of SVT tokens in the existing "svt" Header Parameter.

C.2. JWS Signature SVT Claims

C.2.1. JWS Profile Identifier

When this profile is used, the SigValidation object **MUST** contain a "profile" claim with the value "JWS".

C.2.2. JWS Signature Reference Data

The SVT Signature object **MUST** contain a "sig_ref" claim (SigReference object) with the following elements:

"sig_hash": The hash over the associated signature value (the bytes of the base64url-decoded signature parameter).

"sb_hash": The hash over all bytes signed by the associated signature (the JWS Signing Input according to [RFC7515]).

C.2.3. JWS Signed Data Reference Data

The SVT Signature object **MUST** contain one instance of the "sig_data" claim (SignedData object) with the following elements:

"ref": This parameter **MUST** hold one of the following three possible values:

1. The explicit string value "payload" if the signed JWS Payload is embedded in a "payload" member of the JWS.
2. The explicit string value "detached" if the JWS signs detached payload data without explicit reference.
3. A URI that can be used to identify or fetch the detached Signed Data. The means to determine the URI for the detached Signed Data is outside the scope of this specification.

"hash": The hash over the JWS Payload data bytes (not its base64url-encoded string representation).

C.2.4. JWS Signer Certificate References

The SVT Signature object **MUST** contain a "signer_cert_ref" claim (CertReference object). The "type" parameter of the "signer_cert_ref" claim **MUST** be either "chain" or "chain_hash".

- The "chain" type **MUST** be used when signature validation was performed using one or more certificates where some or all of the certificates in the chain are not present in the target signature.
- The "chain_hash" type **MUST** be used when signature validation was performed using one or more certificates where all of the certificates are present in the target signature JOSE header using the "x5c" Header Parameter.

C.3. SVT JOSE Header

C.3.1. SVT Signing Key Reference

The SVT JOSE header must contain one of the following header parameters in accordance with [RFC7515] for storing a reference to the public key used to verify the signature on the SVT:

"x5c": Holds an X.509 certificate [RFC5280] or a chain of certificates. The certificate holding the public key that verifies the signature on the SVT **MUST** be the first certificate in the chain.

"kid": A key identifier holding the Base64-encoded hash value of the certificate that can verify the signature on the SVT. The hash algorithm **MUST** be the same hash algorithm used when signing the SVT as specified by the "`alg`" Header Parameter.

Appendix D. Schemas

D.1. Concise Data Definition Language (CDDL)

The following informative CDDL [RFC8610] expresses the structure of an SVT token:

```
svt = {
  jti: text
  iss: text
  iat: uint
  ? aud: text / [* text]
  ? exp: uint
  sig_val_claims: SigValClaims
}

SigValClaims = {
  ver: text
  profile: text
  hash_algo: text
  sig: [+ Signature]
  ? ext: Extension
}

Signature = {
  sig_ref: SigReference
  sig_data_ref: [+ SignedDataReference]
  signer_cert_ref: CertReference
  sig_val: [+ PolicyValidation]
  ? time_val: [* TimeValidation]
  ? ext: Extension
}

SigReference = {
  ? id: text / null
  sig_hash: binary-value
  sb_hash: binary-value
}
```

```
SignedDataReference = {
  ref: text
  hash: binary-value
}


CertReference = {
  type: "chain" / "chain_hash"
  ref: [+ text]
}

PolicyValidation = {
  pol: text
  res: "PASSED" / "FAILED" / "INDETERMINATE"
  ? msg: text / null
  ? ext: Extension
}

TimeValidation = {
  "time": uint
  type: text
  iss: text
  ? id: text / null
  ? hash: binary-value / null
  ? val: [* PolicyValidation]
  ? ext: Extension
}


Extension = {
  + text => text
} / null

binary-value = text             ; base64 classic with padding
```

D.2. JSON Schema

The following informative JSON schema describes the syntax of the SVT token payload.

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "Signature Validation Token JSON Schema",
    "description": "Schema defining the payload format for SVTs",
    "type": "object",
    "required": [
        "jti",
        "iss",
        "iat",
        "sig_val_claims"
    ],
    "properties": {
        "jti": {
            "description": "JWT ID",
            "type": "string"
        },
```

```
        "iss": {
            "description": "Issuer",
            "type": "string"
        },
        "iat": {
            "description": "Issued At",
            "type": "integer"
        },
        "aud": {
            "description": "Audience",
            "type": [
                "string",
                "array"
            ],
            "items": {"type": "string"}
        },
        "exp": {
            "description": "Expiration time (seconds since epoch)",
            "type": "integer"
        },
        "sig_val_claims": {
            "description": "Signature validation claims",
            "type": "object",
            "required": [
                "ver",
                "profile",
                "hash_algo",
                "sig"
            ],
            "properties": {
                "ver": {
                    "description": "Version",
                    "type": "string"
                },
                "profile": {
                    "description": "Implementation profile",
                    "type": "string"
                },
                "hash_algo": {
                    "description": "Hash algorithm URI",
                    "type": "string"
                },
                "sig": {
                    "description": "Validated signatures",
                    "type": "array",
                    "items": {
                        "$ref": "#/$def/Signature"
                    },
                    "minItems": 1
                },
                "ext": {
                    "description": "Extension map",
                    "$ref": "#/$def/Extension"
                }
            },
            "additionalProperties": false
        }
    },
```

```
"additionalProperties": false,
"$def": {
         "Signature":{
             "type": "object",
             "required": [
                 "sig_ref",
                 "sig_data_ref",
                 "signer_cert_ref",
                 "sig_val"
             ],
             "properties": {
                 "sig_ref": {
                     "description": "Signature Reference",
                     "$ref": "#/$def/SigReference"
                 },
                 "sig_data_ref": {
                     "description": "Signed data array",
                     "type": "array",
                     "items": {
                         "$ref" : "#/$def/SignedDataReference"
                     },
                     "minItems": 1
                 },
                 "signer_cert_ref": {
                     "description": "Signer certificate reference",
                     "$ref": "#/$def/CertReference"
                 },
                 "sig_val": {
                     "description": "Signature validation results",
                     "type": "array",
                     "items": {
                         "$ref": "#/$def/PolicyValidation"
                     },
                     "minItems": 1
                 },
                 "time_val": {
                     "description": "Time validations",
                     "type": "array",
                     "items": {
                         "$ref": "#/$def/TimeValidation"
                     }
                 },
                "ext": {
                    "description": "Extension map",
                    "$ref": "#/$def/Extension"
                }
             },
             "additionalProperties": false
         },
         "SigReference":{
             "type": "object",
             "required": [
                 "sig_hash",
                 "sb_hash"
             ],
             "properties": {
                 "sig_hash": {
                     "description": "Hash of the signature value",
```

```
                    "type": "string",
                    "format": "base64"
                },
                "sb_hash": {
                    "description": "Hash of the Signed Bytes",
                    "type": "string",
                    "format": "base64"
                },
                "id": {
                    "description": "Signature ID reference",
                    "type": ["string","null"]
                }
            },
           "additionalProperties": false
        },
        "SignedDataReference": {
            "type": "object",
            "required": [
                "ref",
                "hash"
            ],
            "properties": {
                "ref": {
                    "description": "Reference to the signed data",
                    "type": "string"
                },
                "hash": {
                    "description": "Signed data hash",
                    "type": "string",
                    "format": "base64"
                }
            },
           "additionalProperties": false
        },
        "CertReference":{
            "type": "object",
            "required": [
                "type",
                "ref"
            ],
            "properties": {
                "type": {
                    "description": "Type of certificate reference",
                    "type": "string",
                    "enum": ["chain","chain_hash"]
                },
                "ref": {
                    "description": "Certificate reference data",
                    "type": "array",
                    "items": {
                        "type": "string",
                        "format": "base64"
                    },
                    "minItems": 1
                }
            },
           "additionalProperties": false
        },
```

```
          "PolicyValidation":{
              "type": "object",
              "required": [
                  "pol",
                  "res"
              ],
              "properties": {
                  "pol": {
                      "description": "Policy identifier",
                      "type": "string"
                  },
                  "res": {
                      "description": "Signature validation result",
                      "type": "string",
                      "enum": ["PASSED","FAILED","INDETERMINATE"]
                  },
                  "msg": {
                      "description": "Message",
                      "type": ["string","null"]
                  },
                  "ext": {
                     "description": "Extension map",
                     "$ref": "#/$def/Extension"
                  }
              },
             "additionalProperties": false
          },
          "TimeValidation":{
              "type": "object",
              "required": [
                  "time",
                  "type",
                  "iss"
              ],
              "properties": {
                  "time": {
                      "description": "Verified time",
                      "type": "integer"
                  },
                  "type": {
                      "description": "Type of time validation proof",
                      "type": "string"
                  },
                  "iss": {
                      "description": "Issuer of the time proof",
                      "type": "string"
                  },
                  "id": {
                      "description": "Time evidence identifier",
                      "type": ["string","null"]

                  },
                  "hash": {
                      "description": "Hash of time evidence",
                      "type": ["string","null"],
                      "format": "base64"
                  },
                  "val": {
```

```
                    "description": "Validation result",
                    "type": "array",
                    "items": {
                        "$ref": "#/$def/PolicyValidation"
                    }
                },
                "ext": {
                   "description": "Extension map",
                   "$ref": "#/$def/Extension"
                }
            },
            "additionalProperties": false
        },
        "Extension": {
          "description": "Extension map",
          "type": ["object","null"],
          "required": [],
          "additionalProperties": {
              "type": "string"
          }
        }
    }
}
```

Appendix E. Examples

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

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

Signature validation token JWT:

```
eyJraWQiOiJPZW5JKzQzNEpoYnZmRG50ZlZcLzhyT3hHN0ZrdnlqYUtWSmFWcUlG
QlhvaFZoQWU1Zks4YW5vdjFTNjg4cjdLYmFsK2Z2cGFIMWo4aWJnNTJRQnkxUFE9
PSIsInR5cCI6IkpXVCIsImFsZyI6IlJTNTEyIn0.eyJhdWQiOiJodHRwOlwvXC9l
eGFtcGxlLmNvbVwvYXVkaWVuY2UxIiwiaXNzIjoiaHR0cHM6XC9cL3N3ZWRlbmNv
bm5lY3Quc2VcL3ZhbGlkYXRvciIsImlhdCI6MTYwMzQ1ODQyMSwianRpIjoiNGQx
Mzk2ZjFmZjcyOGY0MGQ1MjQwM2I2MWM1NzQ0ODYiLCJzaWdfdmFsX2NsYWltcyI6
eyJzaWciOlt7ImV4dCI6bnVsbCwic2lnX3ZhbCI6W3sibXNnIjoiT0siLCJleHQi
Om51bGwsInJlcyI6IlBBU1NFRCIsInBvbCI6Imh0dHA6XC9cL2lkLnN3ZWRlbmNv
bm5lY3Quc2VcL3N2dFwvc2lndmFsLXBvbGljeVwvdHMtcGtpeFwvMDEifV0sInNp
Z19yZWYiOnsic2lnX2hhc2giOiJ5Y2VQVkxJemRjcEs5N0lZT2hGSWYxbnk3OUht
SUNiU1Z6SWVaTmJpem83ckdJd0hOTjB6WElTeUtHakN2bm9uT2FRR2ZMXC9QM3ZE
dEI4OHlLU1dlWGc9PSIsImlkIjoiaWQtNzM5ODljNmZjMDYzNjM2YWI1ZTc1M2Yx
MGY3NTc0NjciLCJzYl9oYXNoIjoiQm9QVjRXQ0E5c0FJYWhqSzFIYWpmRnhpK0F6
QzRKR1R1ZjM5VzNaV2pjekRDVVJ4ZGM5WWV0ZUh0Y3hHVmVnZ3B4SEo3NVwvY1E3
SE4xZERkbGl5SXdnPT0ifSwic2lnbmVyX2NlcnRfcmVmIjp7InJlZiI6WyIxK2Fh
SmV0ZzdyZWxFUmxVRFlFaVU0WklaaFQ0UlV2aUlRWnVLN28xR0ZLYVRQUTZ5K2t4
XC9QTnREcnB1cVE2WGZya0g5d1lESzRleTB5NFdyTkVybnc9PSIsImg0UER4YjVa
S214MWVUU3F2VnZZRzhnMzNzMDVKendCK05nRUhGVTRnYzl0cUcwa2dIa2Y2VzNv
THprVHd3dXJJaDZZOUFhZlpZcWMyelAycEUycDRRPT0iLCJEZDJDNXNCMElPUWVN
Vm5FQmtNNVE5Vzk2bUJITnd3YTJ0elhNcytMd3VZY09VdlBrcnlHUjBhUEc4Tzlu
SVAzbGJ3NktqUTFoRG1SazZ6Qzh4MmpkZz09Il0sInR5cGUiOiJjaGFpbl9oYXNo
In0sInNpZ19kYXRhX3JlZiI6W3sicmVmIjoiIiwiaGFzaCI6IkZjR3BPT2Y4aWxj
UHQyMUdEZDJjR25MR0R4UlM1ajdzdk00YXBwMkg0MWRERUxtMkN6Y2VUWTAybmRl
SmZXamludG1RMzc2SWxYVE9BcjMxeXpZenNnPT0ifSx7InJlZiI6IiN4YWRlcy0x
MWExNTVkOTJiZjU1Nzc0NjEzYmI3YjY2MTQ3N2NmZCIsImhhc2giOiJLUmtnYlo2
UFwvbmhVNjNJTWswR2lVZlVcL0RUd3ZlWWl0ZVFrd0dlSnFDNUJ6VE5WOGJRYnBl
ZFRUdVdKUHhxdkowUlk4NGh3bTdlWVwvZzBIckFPZWdLdz09In1dLCJ0aW1lX3Zh
bCI6W119XSwiZXh0IjpudWxsLCJ2ZXIiOiIxLjAiLCJwcm9maWxlIjoiWE1MIiwi
aGFzaF9hbGdvIjoiaHR0cDpcL1wvd3d3LnczLm9yZ1wvMjAwMVwvMDRcL3htbGVu
YyNzaGE1MTIifX0.TdHCoIUSZj2zMINKg7E44-8VE_mJq6TG1OoPwnYSs_hyUbuX
mrLJpuk8GR5YrndeOucPUYAwPxHt_f68JIQyFTi0agO9VJjn1R7Pj3Jt6WG9pYVN
n5LH-D1maxD11ZxxbcYeHbsstd2Sy2uMa3BdpsstGdPymSmc6GxY5uJoL0-5vwo_
3l-4Bb3LCTiuxYPcmztKIbDy2hEgJ3Hx1K4HF0SHgn3InpqBev3hm2SLw3hH5BCM
rywBAhHYE6OGE0aOJ6ktA5UP0jIIHfaw9i1wIiJtHTaGuvtyWSLk5cshmun9Hkdk
kRTA75bzuq0Iyd0qh070rA8Gje-s4Tw4xzttgKx1KSkvy8n5FqvzWdsZvclCG2mY
Y9rMxh_7607NXcxajAP4yDOoKNs5nm937ULe0vCN8a7WTrFuiaGjry7HhzRM4C5A
qxbDOBXPLyoMr4qn4LRJCHxOeLZ6o3ugvDOOWsyjk3eliyBwDu8qJH7UmyicLxDc
Cr0hUK_kvREqjD2Z
```

Decoded JWT Header:

```
{
  "kid":"OenI+434JhbvfDntfV\/8rOxG7FkvyjaKVJaVqIFBXohVhAe5fK8anov
         1S688r7Kbal+fvpaH1j8ibg52QBy1PQ==",
  "typ":"JWT",
  "alg":"RS512"
}
```

Decoded JWT Claims:

```
{
  "aud" : "http://example.com/audience1",
  "iss" : "https://swedenconnect.se/validator",
  "iat" : 1603458421,
  "jti" : "4d1396f1ff728f40d52403b61c574486",
  "sig_val_claims" : {
    "sig" : [ {
      "ext" : null,
      "sig_val" : [ {
        "msg" : "OK",
        "ext" : null,
        "res" : "PASSED",
        "pol" : "http://id.swedenconnect.se/svt/sigval-policy/
                 ts-pkix/01"
      } ],
      "sig_ref" : {
        "sig_hash" : "ycePVLIzdcpK97IYOhFIf1ny79HmICbSVzIeZNbizo7rGIw
                      HNN0zXISyKGjCvnonOaQGfL/P3vDtB88yKSWeXg==",
        "id" : "id-73989c6fc063636ab5e753f10f757467",
        "sb_hash" : "BoPV4WCA9sAIahjK1HajfFxi+AzC4JGTuf39W3ZWjczDCURx
                     dc9YeteHtcxGVeggpxHJ75/cQ7HN1dDdliyIwg=="
      },
      "signer_cert_ref" : {
        "ref" : [ "1+aaJetg7relERlUDYEiU4ZIZhT4RUviIQZuK7o1GFKaTPQ6y+
                   kx/PNtDrpuqQ6XfrkH9wYDK4ey0y4WrNErnw==",
                  "h4PDxb5ZKmx1eTSqvVvYG8g33s05JzwB+NgEHFU4gc9tqG0kgH
                   kf6W3oLzkTwwurIh6Y9AafZYqc2zP2pE2p4Q==",
                  "Dd2C5sB0IOQeMVnEBkM5Q9W96mBHNwwa2tzXMs+LwuYcOUvPkr
                   yGR0aPG8O9nIP3lbw6KjQ1hDmRk6zC8x2jdg==" ],
        "type" : "chain_hash"
      },
      "sig_data_ref" : [ {
        "ref" : "",
        "hash" : "FcGpOOf8ilcPt21GDd2cGnLGDxRS5j7svM4app2H41dDELm2Czc
                  eTY02ndeJfWjintmQ376IlXTOAr31yzYzsg=="
      }, {
        "ref" : "#xades-11a155d92bf55774613bb7b661477cfd",
        "hash" : "KRkgbZ6P/nhU63IMk0GiUfU/DTwveYiteQkwGeJqC5BzTNV8bQb
                  pedTTuWJPxqvJ0RY84hwm7eY/g0HrAOegKw=="
      } ],
      "time_val" : [ ]
    } ],
    "ext" : null,
    "ver" : "1.0",
    "profile" : "XML",
    "hash_algo" : "http://www.w3.org/2001/04/xmlenc#sha512"
  }
}
```

Authors' Addresses

Stefan Santesson
IDsec Solutions AB
Forskningsbyn Ideon
SE-223 70 Lund
Sweden
Email: sts@aaa-sec.com

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA 20170
United States of America
Email: housley@vigilsec.com