Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 24 April 2021

S. Santesson
3xA Security
R. Housley
Vigil Security

21 October 2020

Signature Validation Token
draft-santesson-svt-00

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. SVT supports electronic signatures in CMS, XML, and PDF documents.

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 date and time when the signature was actually created is not available.

* Algorithms used to create the electronic signature are no longer considered secure.
* Services necessary to validate the signature are no longer available.
* Supporting evidence such as CA certificates, OCSP responses, CRLs, or timestamps.

The challenges to validation of an electronic signature increases over time, and eventually it is simply 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 prolong the lifetime of a signature by storing data that supports validation of the electronic signature beyond the lifetime of the certificate containing the signer's public key, which is often referred to as the signing certificate. The problem with this approach is that the amount of information that must be stored along with the electronic signature constantly grows over time. The increasing amount of information and signed objects that need to be validated in order to verify the original electronic signature grows in complexity to the point where validation of the electronic signature may become infeasible.

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

This approach drastically reduces the complexity of validation of older electronic signatures for the simple reason that validating the SVT eliminates the need to validate the many signed objects that would otherwise been needed to provide the same level of assurance. 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, 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.
* 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.

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

3. Signature Validation Token

The Signature Validation Token (SVT) is created by a trusted service to capture evidence of successful electronic signature verification, and then relying parties can depend on the checking that has already taken place by the trusted service.

3.1. Signature Validation Token Function

The function of the SVT is to capture evidence of electronic 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 electronic 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 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 that the signature was verified, the date and 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 date and time at which the signature is known to have existed, the procedures used to validate the date and 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 subsequent revalidating the electronic 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 safe and trusted environment to expand beyond a locally controlled environment, and the SVT allows a greater period between original electronic signature verification and subsequent usage.

Using the SVT, the electronic signature verification of a document can be take place once using a reliable trusted service, and then any relying party that is able to depend on the verification process already performed by the trusted service. The SVT is therefore not only a valuable tool to extend the lifetime of a signed document, but also avoids the need for careful integration between electronic signature verification and document usage.

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 of Base64 encoded byte array of binary data.
* StringOrURI - JSON Data Type of string that contains an arbitrary string or an URI as defined in [RFC7519], which REQUIRES a value containing the colon character (":") to be a URI.
* URI - JSON Data Type of string that contains an 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 a 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 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. A 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 an URI based on a domain owned by the issuer. A SVT MUST contain one "Issuer" claim.
* iat - A NumericDate data type that is an "Issued At" registered claim according to [RFC7519], which expresses the date and time when this SVT was issued. A 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 a SVT is OPTIONAL.
* exp - A NumericDate data type that is an "Expiration Time" registered claim according to [RFC7519], which expresses the date and 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. Inclusion of the "Expiration Time" claim in a SVT is OPTIONAL.

* sig_val_claims - A Object<SigValidation> data type that contains signature validation claims for this SVT extending the standard registered JTW claims above. A SVT MUST contain one sig_val_claims claim.

Note: An SVT asserts that a particular validation process was undertaken at a stated date and 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. Inclusion of this parameter is OPTIONAL.
* 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 [RFC6931] or in the IANA registry defined by this specification. This parameter MUST be present.
* sig - A [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 - A Object<SigReference> data type that contains reference information identifying the target signature. This parameter MUST be present.
* sig_data - A [Object<SignedData>] data type that contains an array of references to Signed Data that was signed by the target electronic signature. This parameter MUST be present.
* signer_cert_ref - A Object<CertReference> data type that references the signer's certificate and optionally reference to a supporting certification path that was used to verify the target electronic signature. This parameter MUST be present.
* sig_val - A [Object<PolicyValidation>] data type that contains an array of results of signature verification according to defined procedures. This parameter MUST be present.
* time_val - A [Object<TimeValidation>] data type that contains an array of time verification results that the target signature has existed at a specific date and 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. SignedData Claims Object Class

The sig_data parameter in the Signature object class uses the SignedData object class. The SignedData 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 spefific 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 time evidence asserting that the target signature existed at a particular date and time in the past, and it 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.
* val - A [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. This 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 of the signer.

3.2.10. SVT JOSE Header

The SVT JWT MUST contain the following 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 [add a ref]. 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.

Each profile MUST as a minimum define:

* How to reference the Signed Data content of the signed document.
* How to reference to the target electronic signature and the Signed Bytes of the signature.
* How to reference certificates supporting each electronic siganture.
* How to include public keys or references to public keys in the SVT.
* Whether each electronic signature is supported by a single SVT, or whether 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.

5. Signature Verification with a SVT

Signature verification based on an a 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

{ To be written }

7. Security Considerations

{ To be written }

8. Normative References

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

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

[ISOPDF2] ISO, "Document management -- Portable document format -- Part 2: PDF 2.0", ISO 32000-2, July 2017.

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

[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>.

[RFC6931] Eastlake 3rd, D., "Additional XML Security Uniform Resource Identifiers (URIs)", RFC 6931, DOI 10.17487/RFC6931, April 2013, <https://www.rfc-editor.org/info/rfc6931>.

[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>.

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

[XMLDSIG11] Eastlake, 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, 11 April 2013.

Appendix A. Appendix: 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 readablilty. Line breaks are not allowed in valid JSON data.

Signature validation token JWT:

eyJraWQiOiJPZW5JKzQzNEpoYnZmRG50ZlZcLzhyT3hHN0ZrdnlqYUtWSmFWcUlG
QlhvaFZoQWU1Zks4YW5vdjFTNjg4cjdLYmFsK2Z2cGFIMWo4aWJnNTJRQnkxUFE9
PSIsInR5cCI6IkpXVCIsImFsZyI6IlJTNTEyIn0.eyJhdWQiOiJodHRwOlwvXC9l
eGFtcGxlLmNvbVwvYXVkaWVuY2UxIiwiaXNzIjoiaHR0cHM6XC9cL3N3ZWRlbmNv
bm5lY3Quc2VcL3ZhbGlkYXRvciIsImlhdCI6MTU4MjczMDY0NSwianRpIjoiZTIy
YzViZTZkZDZjYzZkYjgzNGJjY2QwNjZmNWUyZTMiLCJzaWdfdmFsX2NsYWltcyI6
eyJzaWciOlt7ImV4dCI6bnVsbCwic2lnX3ZhbCI6W3sibXNnIjoiSW52YWxpZCBz
aWduYXR1cmUiLCJleHQiOm51bGwsInJlcyI6IkZBSUxFRCIsInBvbCI6Imh0dHA6
XC9cL2lkLnN3ZWRlbmNvbm5lY3Quc2VcL3N2dFwvc2lndmFsLXBvbGljeVwvY2hh
aW5cLzAxIn1dLCJzaWdfcmVmIjp7InNpZ19oYXNoIjoiQmh1RTlCQ1lkcUxneW93
bDJQYm1uSzlkSkFtaVZ0VDF1OVZnaUY5OWgyaFZQekU0WExXdmJDUGU0YUNKM0l6
RmZvTDlrM3RXcjBXK3d5OUJlcWlyY1E9PSIsImlkIjpudWxsLCJzYl9oYXNoIjoi
YnVlcTVIVE8xYnRwQ3JYUlg3VHpFS1VyTkpRaEdHOHFCaDR3eEVTcVJMM0J6bjRj
bHZLMzdqWXUwS2pNTWtnSlFFTWZBMWIzaW1peTc5dDdoK1loOHc9PSJ9LCJzaWdu
ZXJfY2VydF9yZWYiOnsicmVmIjpbIk5TdUZNXC92SitiZUJsUXRRVHptY1loNXg3
TDhXQzlFMUtQSFJBMWlvTk9sS1ZHYmxhOVVSelljc2lzQXgyYmNzcU9oa3ZWVGMz
bUs5RTZhZzA3aGZhdz09Il0sInR5cGUiOiJjaGFpbl9oYXNoIn0sInNpZ19kYXRh
X3JlZiI6W3sicmVmIjoiMCAxMjI5MzUgMTI3OTM3IDI3NDMwIiwiaGFzaCI6Imt1
VWI4NkZzTU5tSmwzdjRiUUswOUZrUWd2bzlReDAxbk5SeVFLVVppaEdFdW1kVnF0
dUJLTlBxWkkxVHpDUWV3Nm44b0ZNak5oQjhDMFhNSmxrRE9RPT0ifV0sInRpbWVf
dmFsIjpbXX1dLCJleHQiOnsibmFtZTIiOiJ2YWwyIiwibmFtZTEiOiJ2YWwxIn0s
InZlciI6IjEuMCIsInByb2ZpbGUiOiJQREYiLCJoYXNoX2FsZ28iOiJodHRwOlwv
XC93d3cudzMub3JnXC8yMDAxXC8wNFwveG1sZW5jI3NoYTUxMiJ9fQ.DhrCRxT_U
8LeqK1BU9-5Bqui2cs5n21PrSqPnDtVa7mxUtqTnouOXjVfuWR0lfNAjEkc1y2QS
X5x2dmMdCpNLWX127UHYiAm8NzeYuoWqdnxKiy61hZ1l0Jldnk52ngG_2UNDnrCG
Bo9OgC90kG2bFQimZB3WgVtE7ad_HAwIXwd-vEHt6Sf2yWXlUTYqQ1Dxgq6pTKQn
uf5ahsHVyeDihgNeix8-cGx1MEvvHNUpCcIXBx67BEcZ-SrqRoIZkVqEcW83KFMg
qKWmWDgp4z_CKM5ix2dVzwp1GvYOM6M3QUKYgmiNA6dMWJvXeJZ-KKi5A-6gEqfg
OsixuZechcDon_3nMzEeNBSJFXU7ohkvxIJN9LXNFAxzAT2UmASxrL9wvaQMmJHY
Meet-vUsOPWcsq07eKO5bnsYwrs9igYeotgcT_Nl74Rmf9uMye_IgyzlS_NLL4xq
9Aaf6LPXWZ0S_plugvfzv7HuzXNOY994voq8sOpO9xKYhqYnzbdDFKU

Decoded JWT Header:

{
  "kid" : "OenI+434JhbvfDntfV\/8rOxG7FkvyjaKVJaVqIFBXohVhAe5fK8an
           ov1S688r7Kbal+fvpaH1j8ibg52QBy1PQ==",
  "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" : "BhuE9BCYdqLgyowl2PbmnK9dJAmiVtT1u9VgiF99h2h
                             VPzE4XLWvbCPe4aCJ3IzFfoL9k3tWr0W+wy9BeqircQ
                             ==",
               "id" : null,
               "sb_hash" : "bueq5HTO1btpCrXRX7TzEKUrNJQhGG8qBh4wxESqRL3B
                            Bzn4clvK37jYu0KjMMkgJQEMfA1b3imiy79t7h+Yh8w=
                            ="
             },
             "signer_cert_ref" : {
               "ref" : [ "NSuFM/vJ+beBlQtQTzmcYh5x7L8WC9E1KPHRA1ioNOlKVG
                          bla9URzYcsisAx2bcsqOhkvVTc3mK9E6ag07hfaw==" ],
               "type" : "chain_hash"
             },
             "sig_data_ref" : [ {
               "ref" : "0 122935 127937 27430",
               "hash" : "kuUb86FsMNmJl3v4bQK09FkQgvo9Qx01nNRyQKUZihGEumd
                         VqtuBKNPqZI1TzCQew6n8oFMjNhB8C0XMJlkDOQ=="
             } ],
             "time_val" : [ ]
           } ],
           "ext" : {
             "name2" : "val2",
             "name1" : "val1"
           },
           "ver" : "1.0",
           "profile" : "PDF",
           "hash_algo" : "http://www.w3.org/2001/04/xmlenc#sha512"
         }
       }
```


Authors' Addresses

Stefan Santesson
3xA Security 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