Javascript Object Signing and Encryption
Internet-Draft
Intended status: Informational
Expires: 8 January 2026

P. Bastian
M. Kraus
Bundesdruckerei GmbH
S. Santesson
IDsec Solutions
P. L. Altmann
The Agency for Digital Government

7 July 2025

Designated Verifier Signatures for JOSE
draft-bastian-jose-dvs-01

Abstract

This specification defines structures and algorithm descriptions for the use of designated verifier signatures, based on a combination of Key Agreement and Message Authentication Code, with JOSE.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://paulbastian.github.io/draft-bastian-jose-dvs/draft-bastian-jose-dvs.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-bastian-jose-dvs/.

Discussion of this document takes place on the Javascript Object Signing and Encryption Working Group mailing list (mailto:jose@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/jose/. Subscribe at https://www.ietf.org/mailman/listinfo/jose/.

Source for this draft and an issue tracker can be found at https://github.com/paulbastian/draft-bastian-jose-dvs.

Status of This Memo

This Internet-Draft will expire on 8 January 2026.

Copyright Notice

Table of Contents

1. Introduction

Designated verifier signatures (DVS) are signature schemes in which signatures are generated, that can only be verified by a particular party. Unlike conventional digital signature schemes like ECDSA, this enables repudiable signatures.

This specification describes a general structure for designated verifier signature schemes and specified a set of instantiations that use a combination of an KA-DH (Diffie-Hellman key aggrement) with an MAC (Message Authentication Code algorithm).

The combination of ECKA-DH and MAC is a established mechanism and used, for example, in the mobile driving licence (mDL) application, specified in [ISO-18013-5].

This specification and all described algorithms should respect the efforts for Fully Specified Algorithms (https://www.ietf.org/archive/id/draft-jones-jose-fully-specified-algorithms-00.html).

This algorithm is intended for use with digital credentials ecosystems, including the Issuer-Holder-Verifier model described by W3C VCDM or IETF SD-JWT-VC.

2. Conventions and 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.

3. Terminology

The draft uses "JSON Web Signature", "JOSE Header", "JWS Signature", "JWS Signing Input" as defined by [RFC7515].

Signing Party: The Party that performs the key agreement first and generates the MAC. Similar to a Signer.

Verifying Party: The Party that performs the key agreement second, generates the MAC and compares it to a given value. Similar to a Verifier.

4. Cryptographic Dependencies

DVS rely on the following primitives:

* A Diffie-Hellman Key Agreement (KA-DH), for example ECKA-DH defined in [BSI-TR-03111]:
  - DH(skX, pkY): Perform a non-interactive Diffie-Hellman exchange using the private key skX and public key pkY to produce a Diffie-Hellman shared secret of length Ndh. This function can raise a ValidationError.
  - Ndh: The length in bytes of a Diffie-Hellman shared secret produced by DH().
  - Nsk: The length in bytes of a Diffie-Hellman private key.
* A key derivation function (KDF), for example HKDF defined in [RFC5869]:
  - Extract(salt, ikm): Extract a pseudorandom key of fixed length Nh bytes from input keying material ikm and an optional byte string salt.
  - Expand(prk, info, L): Expand a pseudorandom key prk using optional string info into L bytes of output keying material.
  - Nh: The output size of the Extract() function in bytes.
* A Message Authentication Code algorithm (MAC), for example HMAC defined in [RFC2104]:
  - MacSign(k, i): Returns an authenticated tag for the given input i and key k.
  - Nk: The length in bytes of key k.

5. Designated Verifier Signatures

A designated verifier signature requires three components for an algorithm:

1. a Diffie-Hellman Key Agreement (DHKA)
2. a Key Derivation Function (KDF)
3. a Message Authentication Code algorithm (MAC)

In general, these parameters are chosen by the Signing Party. These parameters need to be communicated to the Verifying Party before the generation of a Designated Verifier Signature.

5.1. Signature Generation

The generation of the Designated Verifier Signature takes the private key of the Signing Party, the public key of the Verifying Party and the message as inputs. The retrieval and communication of the Verifying Party’s public key is out of scope of this specification and subject to the implementing protocols.

Input:

* skS: private key of the Signing Party

* pkR: public key of the Verifying Party

* msg: JWS Signing Input

* salt : Salt for key derivation

* info : optional info for key derivation

Function:

```
def dvsSign(skS, pkR, msg, salt= "", info = "DVS-1")

    dh =  DH(skS, pkR)
    prk = Extract(salt, dh)
    k = Expand(prk, info, Nk)
    signature = MacSign(k, msg)
    return signature
```

5.2. Signature Verification

The generation of the Designated Verifier Signature takes the private key of the Signing Party, the public key of the Verifying Party and the message as inputs. The retrieval and communication of the Verifying Party’s public key is out of scope of this specification and subject to the implementing protocols.

Input:

* skR: private key of the Verifying Party

* pkS: public key of the Signing Party

* msg: JWS Signing Input

* salt : Salt for key derivation

* info : optional info for key derivation

* signature : the Message Authentication Code

Function:

```
def dvsVerify(skR, pkS, msg, salt = "", info = "DVS-1", signature)

    dh =  DH(skR, pkS)
    prk = Extract(salt, dh)
    k = Expand(prk, info, Nk)
    signature’ = MacSign(k, msg)
    if signature != signature’:
    raise Exception("Designated Verifier Signature invalid")
    return
```

5.3. Signature Suites

Algorithms MUST follow the naming DVS-<DHKA>-<KDF>-<MAC>.

6. Designated Verifier Signatures for JOSE

Designated Verifier Signatures behave like a digital signature as described in Section 3 of [RFC7518] and are intended for use in JSON Web Signatures (JWS) as described in [RFC7515]. The Generating Party performs the Message Signature or MAC Computation as defined by Section 5.1 of [RFC7515]. The Verifying Party performs the Message Signature or MAC Validation as defined by Section 5.2 of [RFC7515].

The following JWS headers are used to convey Designated Verifier Signatures for JOSE:

* alg : REQUIRED. The algorithm parameter describes the chosen signature suite, for example the ones described in (#generic_suites).

* rpk : REQUIRED. The rpk (recipient public key) parameter represents the encoded public key of the Verifying Party that was used in the DHKA algorithm as a JSON Web Key according to [RFC7517]. This parameter MUST be present.

* nonce : OPTIONAL. The nonce may be provided by the Verifying Party additional to it's public key and ensure additional freshness of the signature. If provided, the Signing Party SHOULD add the nonce to the header.

The Signing Party may use existing JWS header parameters like x5c, jwk or kid to represent or reference it's public key according to [RFC7517].

6.1. Example JWT

The JWT/JWS header:

```
{
    "typ" : "JWT",
    "alg" : "DVS-P256-SHA256-HS256",
    "jwk" : <JWK of the Signing Party>,
    "rpk" : <JWK of Verifying Party>
}
```

The JWT/JWS payload:

```
{
    "iss" : "https://example.as.com",
    "iat" : "1701870613",
    "given_name" : "Erika",
    "family_name" : "Mustermann"
}
```

The JWT/JWS signature:

base64-encoded MAC

This specification described instantiations of Designated Verifier Signatures using specific algorithm combinations:

Algorithm Name	Algorithm Description	Requirements
DVS-P256-SHA256-HS256	ECDH using NIST P-256, HKDF using SHA-256 and HMAC using SHA-256	Optional

7. Security Considerations

7.1. Replay Attack Detection

Verifying party MUST ensure the freshness of signatures by utilizing ephemeral keys in rpk or by providing a nonce for nonce.

7.2. Limited Repudiability

A malicious verifiying party can weaken the repudiability property by involving certain third parties in the protocol steps.

* One method is to have a third party observe all protocol steps so that third party can be sure that the signature originates by the signer.

* Another method requires that the verifying party’s public key is a shared key that has previously been calculated with the keys of certain specific third parties so that the proof of authenticity can be done with Multi Party Computation involving all parties (see [TLS-NOTARY]).

8. IANA Considerations

Define:

* define new rpk header parameter

* alg values for DVS-P256-SHA256-HS256 and some more

9. References

9.1. Normative References

[BSI-TR-03111] "Technical Guideline BSI TR-03111: Elliptic Curve Cryptography, Version 2.10", June 2018, <https://www.bsi.bund.de/dok/TR-03111-en>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/rfc/rfc2104>.

[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/rfc/rfc2119>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[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/rfc/rfc7515>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

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

[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/rfc/rfc8174>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/rfc/rfc9180>.

9.2. Informative References

[ISO-18013-5] "ISO/IEC 18013-5:2021, Personal identification — ISO-compliant driving licence, Part 5: Mobile driving licence (mDL) application", September 2021, <https://www.iso.org/standard/69084.html>.

[TLS-NOTARY] "TLSNotary project", October 2024, <https://tlsnotary.org/>.

Appendix A. Acknowledgments

Thanks to:

* Brian Campbell

* John Bradley

Authors' Addresses

Paul Bastian
Bundesdruckerei GmbH
Email: bastianpaul@googlemail.com

Micha Kraus
Bundesdruckerei GmbH
Email: kraus.micha@gmail.com

Stefan Santesson
IDsec Solutions
Email: stefan@aaa-sec.com

Peter Lee Altmann
The Agency for Digital Government
Email: altmann@mail.com