Javascript Object Signing and Encryption
Internet-Draft
Intended status: Informational
Expires: 15 April 2026

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

12 October 2025

Public Key Derived HMAC for JOSE
draft-bastian-jose-dvs-02

Abstract

This specification defines the use of a Diffie-Hellman key agreement (DH-KA) protocol combined with a key derivation function (KDF) to derive a symmetric Message Authentication Code (MAC) key from public information conveyed within a JSON Web Signature (JWS).

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 is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

JSON Web Signature (JWS) [RFC7515] and JSON Web Algorithms (JWA) [RFC7518] specify how to secure content with Hash-based Message Authentication Codes (HMAC) [RFC2104] using a shared symmetric key. These specifications do not provide means to dynamically derive a MAC key for JWS validation using only public information embedded in the JWS.

This specification defines a new protected header parameter, pkds (public key derived secret), which contains information required to derive an HMAC key using a Diffie-Hellman key agreement (DH-KA) and a key derivation function (KDF). The JWS Producer's DH-KA public key appears either in the pkds parameter or in a claims element for use in the key agreement computation. The pkds parameter also includes the JWS Recipient's DH-KA public key, used by the JWS Producer during key agreement, as well as the KDF parameters necessary for deriving the MAC key.

This specification also defines new alg parameter values, that are fully-specified according to Fully Specified Algorithms (https://www.ietf.org/archive/id/draft-jones-jose-fully-specified-algorithms-00.html).

The method is useful in settings where pre-shared keys are undesirable or infeasible, and where direct key distribution or key wrapping introduces operational concerns. It enables the use of HMAC-based signatures that can be validated solely with information embedded in a JWS.

A primary motivation for this work is to enable HMAC signature validation from information contained within an SD-JWT, mirroring capabilities available in credential formats like [ISO-18013-5].

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].

Producer: The party that performs the DH-KA first, derives the MAC

key via a KDF, constructs the JOSE Header and JWS Payload, and computes the JWS Signature.

Recipient: The party that performs the DH-KA second, derives the MAC key via information in the JWS, and validates the JWS using the MAC key according to [RFC7515].

4. Cryptographic Dependencies

This specification relies 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. Public Key Derived HMAC

A public key derived HMAC 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 Producer. These parameters need to be communicated to the Recipient to be able to verify the HMAC.

5.1. Signature Generation

The generation of the public key derived HMAC uses the Producer' s private key, the Recipient' s public key, and the message as inputs, along with optional parameters. The retrieval and communication of the Recipient's public key is out of scope of this specification and subject to the implementing protocols.

Input:

* skP: private key of the Producer

* pkR: public key of the Recipient

* msg: JWS Signing Input

* salt : Optional salt for key derivation

* info : Optional info for key derivation

Function:

```
def pkdHhmacSign(skP, pkR, msg, salt, info)

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

5.2. Signature Verification

The generation of the public key derived HMAC uses the Recipient's private key, the Producer's public key, and the message as inputs, along with optional parameters.

Input:

* skR: private key of the Recipient

* pkS: public key of the Producer

* msg: JWS Signing Input

* salt : Optional salt for key derivation

* info : Optional info for key derivation

* signature : the Message Authentication Code

Function:

```
def pkdHhmacVerify(skR, pkS, msg, signature, salt, info)

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

5.3. Signature Suites

Algorithms MUST follow the naming **TODO**.

6. Public Key Derived HMAC for JOSE

Public Key Derived HMAC 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 Producer performs the Message Signature or MAC Computation as defined by Section 5.1 of [RFC7515]. The Recipient performs the Message Signature or MAC Validation as defined by Section 5.2 of [RFC7515].

The following JWS headers are used to convey Public Key Derived HMAC for JOSE:

* alg : REQUIRED. The algorithm parameter describes the chosen signature suite, for example the ones described in Suites (Section 5.3).

* pkds : REQUIRED. The pkds (Public key derived secret) parameter specifies the inputs needed to derive a symmetric key for MAC PKDS Headermake (Section 6.1).

* nonce : OPTIONAL. The nonce may be provided by the Recipient additional to it’s public key and ensure additional freshness of the signature. If provided, the Producer SHOULD add the nonce to the header.

6.1. The "pkds" Header Parameter

The pkds protected header parameter specifies the inputs needed to derive a symmetric key for MAC computation using a key agreement and derivation scheme. Its value is a JSON object that includes identifiers, public keys, and algorithm-specific parameters relevant to the derivation.

6.1.1. Syntax and semantics

The pkds Header Parameter value MUST be a JSON object with the following fields:

* rpk (object, REQUIRED): The Recipient’s public key used in DH-KA. The rpk object MUST contain at least one key claim as defined in Section 4.1 of [RFC7515].

  Implementations MUST reject a JWS if the rpk key cannot be resolved unambiguously at validation time.

* ppk (object, OPTIONAL): The JWS Producer’ s public key used in DH-KA. The ppk object MUST contain at least one key claim as defined in Section 4.1 of [RFC7515].

  Implementations MUST reject a JWS if the ppk key cannot be resolved unambiguously at validation time or is incompatible with the key information in rpk.

* info (string, OPTIONAL): Context- and application-specific information used as the info parameter to the KDF.

For a machine-readable definition of these fields, see the JSON Schema in Appendix A (Appendix B).

6.2. Example JWT (*TODO*)

The JWT/JWS header:

```
{
  "typ":"JWT",
  "alg":"ECDH-P256+HKDF-SHA256+HS256",
  "pkds":{
    "pkS":{
      "jwk":{
        "kty":"EC",
        "crv":"P-256",
        "x":"f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU",
        "y":"x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0"
      }
    },
    "pkR":{
      "jwk":{
        "kty":"EC",
        "crv":"P-256",
        "x":"vWV4vmRT8HV9QAkbZJAWrvjOV0NfNOzkzq92apq8yjk",
        "y":"XGJRplhubKDv6cFA7e9-yn7F89UUwt57JVLAOS1tpXE"
      }
    },
    "info":"PKDS-v1"
  }
}
```


The JWT/JWS payload (TODO!:


```
{
  "sub": "1234567890",
  "iat": 1516239022
}
```


The JWT/JWS signature:

base64-encoded MAC

This specification described instantiations of Public Key Derived HMAC using specific algorithm combinations:

Algorithm Name	Algorithm Description	Requirements
"ECDH-P256+HKDF-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

Recipient 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 Recipient 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 Producer.

* Another method requires that the Recipient’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 pkds header parameter

* alg values for *TODO* 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

Appendix B. Appendix A. JSON Schema for the "pkds" Header Parameter

JSON { "$schema": "https://json-schema.org/draft/2020-12/schema", "$id": "https://example.com/schemas/pkds.schema.json", "title": "JOSE Header Parameter: pkds", "type": "object", "properties": { "ppk": { "$ref": "#/$defs/keyRef" }, "rpk": { "$ref": "#/$defs/keyRef" }, "params": { "type": "object" } }, "required": ["rpk"], "additionalProperties": false, "$defs": { "keyRef": { "type": "object", "properties": { "jwk": { "type": "object" }, "kid": { "type": "string" }, "jkt": { "type": "string" }, "jku": { "type": "string" }, "x5c": { "type": "array", "items": { "type": "string" } }, "x5u": { "type": "string" }, "x5t": { "type": "string" } }, "anyOf": [{ "required": ["jwk"] }, { "required": ["kid"] }, { "required": ["jkt"] }, { "required": ["jku"] }, { "required": ["x5c"] }, { "required": ["x5u"] }, { "required": ["x5t"] }], "additionalProperties": false } } }

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