COSE
Internet-Draft
Intended status: Standards Track
Expires: 31 August 2026

E. Lundberg, Ed.
Yubico
M. B. Jones
Self-Issued Consulting

27 February 2026

Split signing algorithms for COSE
draft-lundberg-cose-two-party-signing-algs-06

Abstract

This specification defines COSE algorithm identifiers for negotiating how to split a signature algorithm between two cooperating parties. Typically the first party hashes the data to be signed and the second party finishes the signature over the hashed data. This is a common technique, useful for example when the signing private key is held in a smart card or similar hardware component with limited processing power and communication bandwidth. The resulting signatures are identical in structure to those computed by a single party, and can be verified using the same verification algorithm without additional steps to preprocess the signed data.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-lundberg-cose-two-party-signing-algs/.

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

Source for this draft and an issue tracker can be found at https://github.com/YubicoLabs/cose-two-party-signing-algs-rfc.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

CBOR Object Signing and Encryption (COSE) [RFC9052] algorithm identifiers are used for algorithm negotiation and to annotate cryptographic objects with how to interpret them, for example which algorithm to use to verify a signature or decapsulate a shared key. Existing COSE algorithm identifiers omit some internal details of how the object was constructed, since those details are typically irrelevant for the recipient.

The algorithm identifiers defined in this specification are meant for a complementary use case: to divide responsibilities during _construction_ of a cryptographic object, instead of describing how to _consume_ the object. Specifically, they provide an interoperable way to negotiate how a signing operation is split between two cooperating parties, for example, a smart card and a software application, while the verification algorithm for the resulting signature remains the same as if the signature was created by a single party. These split algorithm identifiers are therefore not meant for annotating signature objects, since the verification algorithm is better indicated using already existing algorithm identifiers.

As mentioned above, a primary use case for this is for algorithm negotiation between a software application and a smart card or other hardware security module (HSM) holding the signing private key. Since the HSM may have limited processing power and communication bandwidth, it may not be practical to send the entire original message to the HSM. Instead, since most signature algorithms begin with digesting the message into a fixed-length intermediate input, this initial digest can be computed by the software application while the HSM performs the rest of the signature algorithm on the digest. This is a common technique used in standards such as OpenPGP [OPENPGPCARD], PKCS #11 [PKCS11-Spec-v3.1], and PIV [NIST-SP-800-73-5].

Since different signature algorithms digest the message in different ways and at different stages of the algorithm, it is not possible for a cryptographic API to specify that, for example, "the hash digest is computed by the caller" generically for all algorithms. Security considerations for this split may also differ between algorithms.

Instead, the algorithm identifiers defined in this specification enable the parties of that cryptographic API to signal precisely, for each signature algorithm individually, which steps of the algorithm are performed by which party. We thus define two roles: the _digester_ (e.g., a software application) that initializes the signing procedure, and the _signer_ (e.g., an HSM) that holds exclusive control of the signing private key.

Note that these algorithm identifiers do not define new "pre-hashed" variants of the base signature algorithm, nor an intermediate "hash envelope" data structure, such as that defined in [I-D.COSE-Hash-Envelope]. Rather, these identifiers denote existing signature algorithms that would typically be executed by a single party, but split into two stages.

Some signature algorithms, such as PureEdDSA [RFC8032], by their design, cannot be split in this way, and therefore cannot be assigned split signing algorithm identifiers. However, if such a signature algorithm defines a "pre-hashed" variant, that "pre-hashed" algorithm can be assigned a split signing algorithm identifier, enabling the pre-hashing step to be performed by the _digester_ and the remaining steps by the _signer_. For example, this specification defines Ed25519ph-split (Section 2.2) as a split variant of Ed25519ph [RFC8032]. Note that Ed25519 and Ed25519ph have distinct verification algorithms, but Ed25519ph and Ed25519ph-split use the same verification algorithm.

1.1. Requirements Notation and Conventions

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.

1.2. Prior Art

Split signing is a common technique used in existing smart card standards. The following subsections expand on how the technique is applied in OpenPGP [OPENPGPCARD], PKCS #11 [PKCS11-Spec-v3.1], and PIV [NIST-SP-800-73-5].

1.2.1. OpenPGP

The OpenPGP smart card protocol [OPENPGPCARD] defines the format of signing commands in section "7.2.10 PSO: COMPUTE DIGITAL SIGNATURE":

> *7.2.10 PSO: COMPUTE DIGITAL SIGNATURE*

> The command for digital signature computation is shown in the table below. The hash value (ECDSA) or the DigestInfo is delivered in the data field of the command. [...]

The "Data field" parameter is subsequently defined as "Data to be integrated in the DSI: hash value (ELC) or DigestInfo (RSA)". Thus both ECDSA and RSA signatures are computed jointly by the host computing the digest of the signed data and the smart card finalizing the signature on the digest; the host acts as _digester_ and the smart card acts as _signer_.

TODO: Spec 3.4.1 only covers ECDSA and RSA, but some implementations also support Ed25519. Identify and include good references for how OpenPGP smart cards handle Ed25519.

1.2.2. PKCS #11

PKCS #11 [PKCS11-Spec-v3.1] defines signing commands in sections "5.13 Signing and MACing functions" and "5.14 Message-based signing and MACing functions". These sections define C_SignInit and C_MessageSignInit functions that both take a pMechanism parameter indicating the signature mechanism. Mechanisms are defined in section "6 Mechanisms", which notably includes the subsections "6.3.12 ECDSA without hashing" and "6.3.13 ECDSA with hashing":

> *6.3.12 ECDSA without hashing*
>
> [...]
>
> The ECDSA without hashing mechanism, denoted *CKM_ECDSA*, is a mechanism for single-part signatures and verification for ECDSA. (This mechanism corresponds only to the part of ECDSA that processes the hash value, which should not be longer than 1024 bits; it does not compute the hash value.)
>
> [...]
>
> *6.3.13 ECDSA with hashing*
>
> [...]

> The ECDSA with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512 mechanism, denoted *CKM_ECDSA_[SHA1|SHA224|SHA256|SHA384|SHA512|SHA3_224|SHA3_256|SHA3_384|SHA3_512]* respectively, is a mechanism for single- and multiple-part signatures and verification for ECDSA. This mechanism computes the entire ECDSA specification, including the hashing with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512 respectively.
>
> [...]

Thus PKCS #11 supports both split signing using the *CKM_ECDSA* mechanism and "non-split" signing using the *CKM_ECDSA_SHA** mechanisms; when using *CKM_ECDSA*, the PKCS #11 caller acts as _host_ and the Cryptoki implementation acts as _signer_.

1.2.3. PIV: FIPS-201, NIST SP 800

NIST Special Publication 800 [NIST-SP-800-73-5] contains technical specifications for the Personal Identity Verification (PIV) standard [FIPS-201], and defines the format of signing commands in section "3.2.4. GENERAL AUTHENTICATE Card Command":

> *3.2.4. GENERAL AUTHENTICATE Card Command* [...]
>
> The GENERAL AUTHENTICATE command SHALL be used with the digital signature key (’9C’) to realize the signing functionality on the PIV client application programming interface. Data to be signed is expected to be hashed off-card. Appendix A.4 illustrates the use of the GENERAL AUTHENTICATE command for signature generation.
>
> [...]

Appendix A.4 gives examples of RSA and ECDSA signature generation commands. For RSA the command needs to be sent in two parts, giving the "Data Field" argument first as "’7C’ L1 { ’82’ ’00’ ’81’ L2 {first part of the PKCS #1 v1.5 or PSS padded message hash value }}" and then "{second and last part of the PKCS #1 v1.5 or PSS padded message hash value}"; for ECDSA the "Data Field" argument is given as "’7C’ L1 { ’82’ ’00’ ’81’ L2 {hash value of message}}".

Thus both ECDSA and RSA signatures are computed jointly by the host computing the digest of the signed data and the smart card finalizing the signature on the digest; the host acts as _digester_ and the smart card acts as _signer_.

2. Split Signing Algorithms

This section defines divisions of signing algorithm steps between a _digester_ and a _signer_ in a split signing protocol, and assigns algorithm identifiers to these algorithm divisions. The _digester_ performs the first part of the split algorithm and does not have access to the signing private key, while the _signer_ performs the second part of the split algorithm and has access to the signing private key. For signing algorithms that format the message to insert domain separation tags, as described in Section 2.2.5 of [RFC9380], this message formatting is also performed by the _signer_.

How the digest, and any related COSE_Sign_Args structure (see Section 3), are transported from _digester_ to _signer_ is out of scope for this specification, but it is expected that the digest will usually be transmitted as the "data to be signed" argument.

The algorithm identifiers defined in this specification with "-split" in their names MAY appear in COSE structures used internally between the _digester_ and the _signer_ in a split signing protocol, but SHOULD NOT appear in COSE structures consumed by signature verifiers. COSE structures consumed by signature verifiers SHOULD instead use the corresponding conventional algorithm identifiers for the verification algorithm. These are listed in the "Verification algorithm" column in the tables defining split signing algorithm identifiers.

The following subsections define an initial set of split signing algorithm identifiers. The last subsection provides guidance for defining additional identifiers beyond this initial set.

2.1. ECDSA

ECDSA [FIPS-186-5] split signing uses the following division between the _digester_ and the _signer_ of the steps of the ECDSA signature generation algorithm [FIPS-186-5]:

* The signing procedure is defined in Section 6.4.1 of [FIPS-186-5].
* The _digester_ performs Step 1 of the signing procedure - hashing the message, producing the value _H_.
* The message input to the _signer_ is the value _H_ defined in the signing procedure.
* The _signer_ resumes the signing procedure from Step 2.

The following algorithm identifiers are defined:

Name	COSE Value	Verification algorithm	Description
ESP256-split	TBD	ESP256	ESP256 split signing as defined in Section 2.1
ESP384-split	TBD	ESP384	ESP384 split signing as defined in Section 2.1
ESP512-split	TBD	ESP512	ESP512 split signing as defined in Section 2.1

Table 1: ECDSA split signing algorithm values.

Note: This is distinct from the similarly named Split-ECDSA (SECDSA) [SECDSA], although SECDSA can be implemented using this split procedure as a component.

2.2. HashEdDSA

Split HashEdDSA [RFC8032] uses the following division between the _digester_ and the _signer_ of the steps of the HashEdDSA signing algorithm [RFC8032]:

* HashEdDSA is a combination of the EdDSA signing procedure and the PureEdDSA signing procedure. The EdDSA signing procedure is defined in the first paragraph of Section 3.3 of [RFC8032]. The PureEdDSA signing procedure is defined in the second paragraph of Section 3.3 of [RFC8032].

* The _digester_ computes the value PH(M) defined in the EdDSA signing procedure.

* The message input to the _signer_ is the value PH(M) defined in the EdDSA signing procedure. This value is represented as M in the PureEdDSA signing procedure.

* The _signer_ executes the PureEdDSA signing procedure, where the value denoted M in the PureEdDSA signing procedure takes the value denoted PH(M) in the EdDSA signing procedure.

PureEdDSA [RFC8032] cannot be divided in this way since such a division would require that the _digester_ has access to the private key.

The following algorithm identifiers are defined:

Name	COSE Value	Verification algorithm	Description
Ed25519ph	TBD	Ed25519ph	EdDSA using the Ed25519ph parameter set in Section 5.1 of [RFC8032]
Ed25519ph-split	TBD	Ed25519ph	EdDSA using the Ed25519ph parameter set in Section 5.1 of [RFC8032] and split as defined in Section 2.2
Ed448ph	TBD	Ed448ph	EdDSA using the Ed448ph parameter set in Section 5.2 of [RFC8032]
Ed448ph-split	TBD	Ed448ph	EdDSA using the Ed448ph parameter set in Section 5.2 of [RFC8032] and split as defined in Section 2.2

Table 2: HashEdDSA algorithm values.

2.3. Defining Split Signing Algorithms

Future definitions of additional split signing algorithm identifiers SHOULD follow the conventions established in Section 2 as far as possible. For example, if a signature algorithm prescribes insertion of domain separation tags in a way that requires processing the entirety of the data to be signed, it might be necessary to delegate the domain separation responsibility to the _digester_. Per the considerations in Section 4.2, split signing algorithm identifiers SHOULD be defined in ways that minimize how much responsibility is delegated to the _digester_.

As a concrete example, consider ML-DSA and HashML-DSA [FIPS-204]. ML-DSA and HashML-DSA prefix the input data with a 0 octet and a 1 octet respectively, which enforces domain separation between ML-DSA and HashML-DSA signatures. Appendix D of [RFC9881] describes a mode of ML-DSA that could be assigned a split signing algorithm identifier where the _digester_ performs Computeμ and the _signer_ performs Signμ . Note that this puts the _digester_ in control of the domain

separation tags; this is necessary if the hash step is not performed by the _signer_. Therefore with this construction, it is the _digester_ that decides whether the signing protocol will produce an ML-DSA signature or a HashML-DSA signature. In contrast, HashML-DSA first hashes the input data alone and then another time with domain separation tags; therefore HashML-DSA can be assigned a split signing algorithm identifier that keeps the _signer_ in control of the domain separation tags and ensures that the signing protocol can only produce HashML-DSA signatures.

3. COSE Signing Arguments

While many signature algorithms take the private key and data to be signed as the only two parameters, some signature algorithms have additional parameters that must also be set. For example, to sign using a key derived by ARKG [I-D.bradleylundberg-ARKG], two additional arguments kh and ctx are needed in ARKG-Derive-Private-Key to derive the signing private key.

While such additional arguments are simple to provide to the API of the signing procedure in a single-party context, in a split signing context these additional arguments also need to be conveyed from the _digester_ to the _signer_. For this purpose, we define a new COSE structure COSE_Sign_Args for "COSE signing arguments". This enables defining a unified, algorithm-agnostic protocol between the _digester_ and the _signer_, rather than requiring a distinct protocol for each signature algorithm for the sake of conveying algorithm-specific parameters.

COSE_Sign_Args is built on a CBOR map. The set of common parameters that can appear in a COSE_Sign_Args can be found in the IANA "COSE Signing Arguments Common Parameters" registry (TODO). Additional parameters defined for specific signing algorithms can be found in the IANA "COSE Signing Arguments Algorithm Parameters" registry (TODO).

The CDDL grammar describing COSE_Sign_Args, using the CDDL fragment defined in Section 1.5 of [RFC9052], is:

```
COSE_Sign_Args = {
    3 ^ => tstr / int,  ; alg
    * label => values,
}
```

3.1. COSE Signing Arguments Common Parameters

This document defines a set of common parameters for a COSE Signing Arguments object. Table 3 provides a summary of the parameters defined in this section.

Name	Label	CBOR Type	Value Registry	Description
alg	3	tstr / int	COSE Algorithms	Signing algorithm to use

Table 3: Common parameters of the COSE_Sign_Args structure.

* alg: This parameter identifies the signing algorithm the additional arguments apply to. The signer MUST verify that this algorithm matches any key usage restrictions set on the key to be used. If the algorithms do not match, then the signature operation MUST be aborted with an error.

Definitions of COSE algorithms MAY define additional algorithm-specific parameters for COSE_Sign_Args.

The following CDDL example conveys additional arguments for signing data using the ESP256-split algorithm (see Section 2.1) and a key derived using ARKG-P256 [I-D.bradleylundberg-ARKG]:

```
{
  3: -65539,   ; alg: ESP256-split with ARKG-P256 (placeholder value)

               ; ARKG-P256 key handle
               ; (HMAC-SHA-256-128 followed by
                  SEC1 uncompressed ECDH public key)
  -1: h'27987995f184a44cfa548d104b0a461d
        0487fc739dbcdabc293ac5469221da91b220e04c681074ec4692a76ffacb9043de
          c2847ea9060fd42da267f66852e63589f0c00dc88f290d660c65a65a50c86361',

               ; ctx argument to ARKG-Derive-Private-Key
  -2: 'ARKG-P256.test vectors',
}
```

4. Security Considerations

4.1. Protocol-Level Trusted Roles

This specification assumes that both the _digester_ and _signer_ roles described in Section 2 are trusted and cooperate honestly. This is because a similar division between "application" and "secure element" typically exists already: even if all steps of the signing algorithm are executed in a trusted secure element, the inputs to the secure element are provided by some outside component such as a software application. If the application can provide any input to be signed, then for all practical purposes it is trusted with possession of any private keys for as long as it is authorized to exercise the secure element. The application can in practice obtain a signature over any chosen message without needing to perform a forgery attack. The same relationship exists between _digester_ and _signer_. Applications that cannot trust an external _digester_ - for example a hardware security device with an integrated secure display of what data is about to be signed - by definition have no use for split signing algorithm identifiers.

Similarly from a verifier’s perspective, these split signing procedures are implementation details. When a signature is generated by a single party, that single party takes on both the _digester_ and the _signer_ roles, and obviously trusts itself to perform the _digester_ role honestly. From the verifier’s perspective, a malicious _digester_ in the split signing model would have the same powers as a malicious signature generator in a single-party signing model. Thus, on the application or protocol level, assuming an honest _digester_ is no more restrictive than assuming an honest signature generator.

4.2. Component-Level Trusted Roles

The reasoning in Section 4.1 does not hold on the component level. A _signer_ implementation MUST NOT assume that the _digester_ implementation it interoperates with is necessarily honest. Split signing algorithms MUST NOT be defined in a way that enables a malicious _digester_ with access to an honest _signer_ to produce forgeries - any that could not be produced by simply requesting a signature over the desired message - or extract secrets from the _signer_.

For example, for ECDSA (Section 2.1), a malicious _digester_ can choose _H_ in such a way that the _signer_ will derive any _digester_-chosen value of _e_, including zero or other potentially problematic values. Fortunately, in this case, this does not enable the _digester_ to extract the signature nonce or private key. It also does not make forgeries easier, since the _digester_ still needs to find a preimage of _e_ for the relevant hash function.

Definitions of other algorithms need to ensure that similar chosen-input attacks do not enable extracting secrets or forging protocol-level messages.

For example, a naively prehashed version of FALCON [FALCON] would allow such a key compromise: A FALCON signature is a value s such that both s and s * h - hash(r || m) are short, where h is the public key and r a randomizer. If the digester gets to query the signer for signatures of arbitrary stand-ins for hash(r || m), they can extract the private key by for example asking for repeated signatures of 0. Therefore, definitions of split signing algorithms need to be reviewed and potentially have security proofs adjusted.

4.3. Incorrect Use of Split Signing Algorithm Identifiers

Section 2 recommends against exposing split signing algorithm identifiers - including those defined in this specification with "-split" in their names - to signature verifiers. For example, they should not appear as the "alg" parameter of a COSE_Key public key sent to a signature verifier. If a split signing algorithm identifier is encountered in an invalid context like this, the recipient SHOULD reject the message with a fatal error.

This is because the purpose of these split signing algorithm identifiers is to enable more flexible production of signatures that can be verified by existing implementations of existing verification algorithms. A prevalence of these identifiers appearing outside the split signing context would defeat this purpose; we therefore recommend such use SHOULD NOT be tolerated.

This recommendation is the opposite of the oft-cited "robustness principle" stated in paragraph 3.9 of [RFC1958]. Implementations MAY choose to instead follow the robustness principle and tolerate incorrect use of split signing algorithm identifiers, instead interpreting the identifier as referencing the defined corresponding verification algorithm. Note however that this is no longer considered a best practice and is likely to harm interoperability [RFC9413].

A verifier’s choice to tolerate or not tolerate incorrect use of split signing algorithm identifiers is expected to not affect security, assuming a split algorithm identifier is interpreted as an alias representing the same verification algorithm as a non-split algorithm identifier.

5. Implementation Considerations

5.1. Using Non-Split Signing Algorithm Identifiers in a Split Signing Protocol

A protocol designed to use split signing algorithm identifiers such as those defined in this specification MAY also allow use of algorithm identifiers that do not represent split signing algorithms. In this case, the _signer_ performs all steps of the signing procedure as usual. For example, if the _signer_ receives a signature request with an the algorithm identifier "ESP256", then the _digester_ passes the input data through unmodified and it is the _signer_ that computes the SHA-256 digest of the input data as defined in the ESP256 definition [RFC9864].

6. IANA Considerations

6.1. COSE Algorithms Registrations

This section registers the following values in the IANA "COSE Algorithms" registry [IANA.COSE]:

* Name: ESP256-split
  - Value: TBD (Requested Assignment -300)
  - Description: ESP256 split signing
  - Capabilities: [kty]
  - Change Controller: IETF
  - Reference: Section 2.1 of this specification
  - Recommended: Yes
* Name: ESP384-split
  - Value: TBD (Requested Assignment -301)
  - Description: ESP384 split signing
  - Capabilities: [kty]
  - Change Controller: IETF
  - Reference: Section 2.1 of this specification
  - Recommended: Yes

* Name: ESP512-split
  - Value: TBD (Requested Assignment -302)
  - Description: ESP512 split signing
  - Capabilities: [kty]
  - Change Controller: IETF
  - Reference: Section 2.1 of this specification
  - Recommended: Yes
* Name: Ed25519ph
  - Value: TBD
  - Description: EdDSA using the Ed25519ph parameter set in Section 5.1 of [RFC8032]
  - Capabilities: [kty]
  - Change Controller: IETF
  - Reference: Section 5.1 of [RFC8032]
  - Recommended: Yes
* Name: Ed25519ph-split
  - Value: TBD (Requested Assignment -303)
  - Description: Ed25519ph split as defined in Section 2.2
  - Capabilities: [kty]
  - Change Controller: IETF
  - Reference: Section 2.2 of this specification
  - Recommended: Yes
* Name: Ed448ph
  - Value: TBD

- Description: EdDSA using the Ed448ph parameter set in Section 5.2 of [RFC8032]
- Capabilities: [kty]
- Change Controller: IETF
- Reference: Section 5.2 of [RFC8032]
- Recommended: Yes

* Name: Ed448ph-split
  - Value: TBD (Requested Assignment -304)
  - Description: Ed448ph split as defined in Section 2.2
  - Capabilities: [kty]
  - Change Controller: IETF
  - Reference: Section 2.2 of this specification
  - Recommended: Yes

6.2. COSE Signing Arguments Common Parameters Registry

TODO

6.3. COSE Signing Arguments Algorithm Parameters Registry

TODO

7. Implementation Status

This section is to be removed from the specification by the RFC Editor before publication as an RFC.

There are currently two known implementations using features defined by this specification:

* wwWallet (https://github.com/wwWallet), an EU Digital Identity pilot project. wwWallet was entered into the "EUDI Wallet Prototypes" competition held by SprinD GmbH (https://www.sprind.org/en/actions/challenges/eudi-wallet-prototypes), and a branch of the wallet was submitted in the competition. The competition entry implements ARKG [I-D.bradleylundberg-ARKG] for efficiently generating single-use hardware-bound holder binding keys.

  The implementation (https://github.com/gunet/funke-s3a-wallet-frontend/blob/stage-3/src/services/keystore.ts) uses the COSE_Key_Ref data structure defined in version 01 of this specification in order to send ARKG inputs to a WebAuthn authenticator, and uses the placeholder value for the experimental split algorithm identifier ESP256-split-ARKG defined in Section 5.2 of [I-D.bradleylundberg-ARKG] to negotiate creation and usage of ARKG-derived keys for signing operations. Thus wwWallet assumes the _digester_ role while the WebAuthn authenticator assumes the _signer_ role.

* Yubico (https://www.yubico.com/), a hardware security key vendor, has produced limited-availability prototypes of their YubiKey product with an ARKG implementation interoperable with wwWallet. The YubiKey implementation uses the COSE_Key_Ref data structure defined in version 01 of this specification to receive ARKG inputs from a WebAuthn Relying Party, and uses the placeholder value for the experimental split algorithm identifier ESP256-split-ARKG defined in Section 5.2 of [I-D.bradleylundberg-ARKG] to negotiate creation and usage of ARKG-derived keys for signing operations. Thus the YubiKey assumes the _signer_ role while the WebAuthn Relying Party assumes the _digester_ role.

Table 4 summarizes implementation status for individual features.

Feature	Defined by	Digester	Signer
ESP256-split	This specification	-	-
ESP384-split	This specification	-	-
ESP512-split	This specification	-	-
Ed25519ph-split	This specification	-	-
Ed448ph-split	This specification	-	-
ESP256-split-ARKG	[I-D.bradleylundberg-ARKG]	wwWallet	Yubico
ESP384-split-ARKG	[I-D.bradleylundberg-ARKG]	-	-
ESP512-split-ARKG	[I-D.bradleylundberg-ARKG]	-	-
COSE_Sign_Args	This specification	wwWallet	Yubico

Table 4: Implementation status of individual features.

7.1. Dependent Specifications

As indicated in the previous section, the Internet-Draft of ARKG [I-D.bradleylundberg-ARKG] extends this specification with definitions for ARKG:

* Section "5.2 COSE algorithms" defines COSE algorithm identifiers ESP256-split-ARKG, ESP384-split-ARKG and ESP512-split-ARKG based on the ECDSA identifiers defined in this specification (Section 2.1).

* Section "5.3 COSE signing arguments" defines a representation for ARKG arguments using the COSE_Sign_Args data structure defined in this specification (Section 3).

8. References

8.1. Normative References

[I-D.bradleylundberg-ARKG]
Lundberg, E. and J. Bradley, "The Asynchronous Remote Key Generation (ARKG) algorithm", Work in Progress, Internet-Draft, draft-bradleylundberg-cfrg-arkg-10, 27 February 2026, <https://datatracker.ietf.org/doc/html/draft-bradleylundberg-cfrg-arkg-10>.

[IANA.COSE]
IANA, "CBOR Object Signing and Encryption (COSE)", n.d., <https://www.iana.org/assignments/cose/>.

[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>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

[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>.

[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/rfc/rfc8610>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9864] Jones, M.B. and O. Steele, "Fully-Specified Algorithms for JSON Object Signing and Encryption (JOSE) and CBOR Object Signing and Encryption (COSE)", RFC 9864, DOI 10.17487/RFC9864, October 2025, <https://www.rfc-editor.org/rfc/rfc9864>.

[SEC1] Certicom Research, "SEC 1: Elliptic Curve Cryptography", May 2009, <https://www.secg.org/sec1-v2.pdf>.

8.2. Informative References

[FALCON] Fouque, P., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., and Z. Zhang, "FALCON: Fast-Fourier Lattice-based Compact Signatures over NTRU", 2017, <https://falcon-sign.info/>.

[FIPS-186-5] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", February 2023, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf>.

[FIPS-201] National Institute of Standards and Technology, "Personal Identity Verification (PIV) of Federal Employees and Contractors", 2022, <https://csrc.nist.gov/pubs/fips/201-3/final>.

[FIPS-204] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard", August 2024, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf>.

[I-D.COSE-Hash-Envelope] Steele, O., Lasker, S., and H. Birkholz, "COSE Hash Envelope", Work in Progress, Internet-Draft, draft-ietf-cose-hash-envelope-10, 15 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-hash-envelope-10>.

[NIST-SP-800-73-5] Ferraiolo, H., Mehta, K., Francomacaro, S., Chandramouli, R., and S. Gupta, "Interfaces for Personal Identity Verification: Part 2 PIV Card Application Card Command Interface", NIST Special Publication (SP) NIST SP 800-73pt2-5, 2024, <https://doi.org/10.6028/NIST.SP.800-73pt2-5>.

[OPENPGPCARD] Pietig, A., "Functional Specification of the OpenPGP application on ISO Smart Card Operating Systems", Version 3.4.1, March 2020, <https://gnupg.org/ftp/specs/OpenPGP-smart-card-application-3.4.1.pdf>.

[PKCS11-Spec-v3.1] Bong, D. and T. Cox, "PKCS #11 Specification Version 3.1.", OASIS Standard, 23 July 2023, <https://docs.oasis-open.org/pkcs11/pkcs11-spec/v3.1/os/pkcs11-spec-v3.1-os.html>. Latest stage:

https://docs.oasis-open.org/pkcs11/pkcs11-spec/v3.1/pkcs11-spec-v3.1.html (https://docs.oasis-open.org/pkcs11/pkcs11- spec/v3.1/pkcs11-spec-v3.1.html).

[RFC1958] Carpenter, B., Ed., "Architectural Principles of the Internet", RFC 1958, DOI 10.17487/RFC1958, June 1996, <https://www.rfc-editor.org/rfc/rfc1958>.

[RFC9380] Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R. S., and C. A. Wood, "Hashing to Elliptic Curves", RFC 9380, DOI 10.17487/RFC9380, August 2023, <https://www.rfc-editor.org/rfc/rfc9380>.

[RFC9413] Thomson, M. and D. Schinazi, "Maintaining Robust Protocols", RFC 9413, DOI 10.17487/RFC9413, June 2023, <https://www.rfc-editor.org/rfc/rfc9413>.

[RFC9881] Massimo, J., Kampanakis, P., Turner, S., and B. E. Westerbaan, "Internet X.509 Public Key Infrastructure -- Algorithm Identifiers for the Module-Lattice-Based Digital Signature Algorithm (ML-DSA)", RFC 9881, DOI 10.17487/RFC9881, October 2025, <https://www.rfc-editor.org/rfc/rfc9881>.

[SECDSA] Verheul, E., "SECDSA: Mobile signing and authentication under classical "sole control"", July 2021, <https://eprint.iacr.org/2021/910>.

Acknowledgements

We would like to thank Ilari Liusvaara, Lucas Prabel, Sophie Schmieg and Falko Strenzke for their reviews of and contributions to this specification.

Document History

-06

* Added "Prior Art" section to Introduction.

-05

* Fixed ESP384-split misspelled as ESP381-split.

* Clarified that non-"-split" alg IDs defined here may be exposed to verifiers.

* Clarified that transport of digest is out of scope, but expected to be passed as data to be signed.

* Added Security Considerations section "Incorrect Use of Split Signing Algorithm Identifiers".

* Added Implementation Considerations section "Using Non-Split Signing Algorithm Identifiers in a Split Signing Protocol".

* Added section "Defining Split Signing Algorithms" with guidance for handling domain separation tags in new definitions.

* Clarified in introduction that Ed25519 and Ed25519ph(-split) have distinct verification algorithms.

* Clarified in section "Protocol-Level Trusted Roles" why digester is necessarily trusted.

* Clarified in section "Component-Level Trusted Roles" that redundant forgeries are acceptable, and added example of key compromise concern for naively hashed FALCON.

-04

* Added Implementation Status section.

-03

* Updated reference to ARKG parameter info renamed to ctx.

* Refined abstract and introduction to emphasize that the central novelty is not split algorithms as a concept, but providing COSE algorithm identifiers for use cases that benefit from such splitting.

* Replaced reference to draft-ietf-jose-fully-specified-algorithms with RFC 9864.

* Added inline definitions of Ed25519ph and Ed448ph registrations, replacing speculative references to registrations that do not exist elsewhere.

* Added missing captions to Tables 1 and 2.

* Added Security Considerations section.

-02

* Renamed document from "COSE Algorithms for Two-Party Signing" to "Split signing algorithms for COSE" and updated introduction and terminology accordingly.

* Dropped definitions for HashML-DSA, as split variants of ML-DSA are being actively discussed in other IETF groups.

* Changed "Base algorithm" heading in definition tables to "Verification algorithm".

* Remodeled COSE_Key_Ref as COSE_Sign_Args.

  - Dropped definitions of reference types for COSE Key Types registry.

-01

* Added IANA registration requests for algorithms defined.

* Updated references and other editorial tweaks.

-00

* Initial individual draft

Authors' Addresses

Emil Lundberg (editor)
Yubico
Gvlegatan 22
Stockholm
Sweden
Email: emil@emlun.se

Michael B. Jones
Self-Issued Consulting
United States
Email: michael_b_jones@hotmail.com
URI: https://self-issued.info/