jose M. Jones
Internet-Draft Self-Issued Consulting
Intended status: Standards Track D. Waite
Expires: 8 May 2026 J. Miller
Ping Identity
4 November 2025

JSON Proof Algorithms
draft-ietf-jose-json-proof-algorithms-12

Abstract

The JSON Proof Algorithms (JPA) specification registers cryptographic algorithms and identifiers to be used with the JSON Web Proof, JSON Web Key (JWK), and COSE specifications. It defines IANA registries for these identifiers.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The JSON Web Proof (JWP) [I-D.ietf-jose-json-web-proof] draft establishes a new secure container format that supports selective disclosure and unlinkability using Zero-Knowledge Proofs (ZKPs) or other cryptographic algorithms.

> Editor’s Note: This draft is still early and incomplete. There will be significant changes to the algorithms as currently defined here. Please do not use any of these definitions or examples for anything except personal experimentation and learning. Contributions and feedback are welcomed at https://github.com/ietf-wg-jose/json-web-proof (https://github.com/ietf-wg-jose/json-web-proof).

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.

The roles of "issuer", "holder", and "verifier" are used as defined by the VC Data Model [VC-DATA-MODEL-2.0]. The term "presentation" is also used as defined by this source, but the term "credential" is avoided in this specification to minimize confusion with other definitions.

3. Terminology

The terms "JSON Web Signature (JWS)", "Base64url Encoding", "Header Parameter", "JOSE Header", "JWS Payload", "JWS Signature", and "JWS Protected Header" are defined by [RFC7515].

The terms "JSON Web Proof (JWP)", "JWP Payload", "JWP Proof", and "JWP Header" are defined by [I-D.ietf-jose-json-web-proof].

These terms are defined by this specification:

Stable Key: An asymmetric key-pair used by an issuer that is also shared via an out-of-band mechanism to a verifier to validate the signature.

Issuer Ephemeral Key: An asymmetric key-pair that is generated for one-time use by an issuer and never stored or used again outside of the creation of a single JWP.

Holder Presentation Key: An asymmetric key-pair that is generated by a holder and used to ensure that a presentation is not able to be replayed by any other party.

4. Background

JWP defines a container binding together a Header, one or more payloads, and a cryptographic proof. It does not define any details about the interactions between an application and the cryptographic libraries that implement proof-supporting algorithms.

Due to the nature of ZKPs, this specification also documents the subtle but important differences in proof algorithms versus those defined by the JSON Web Algorithms [RFC7518]. These differences help support more advanced capabilities such as blinded signatures and predicate proofs.

5. Algorithm Basics

The four principal interactions that every proof algorithm MUST support are issue (#issue), confirm (#confirm), present (#present), and verify (#verify).

5.1. Issue

The JWP is first created as the output of a JPA’s issue operation.

Every algorithm MUST support a JSON issuer Header along with one or more octet string payloads. The algorithm MAY support using additional items provided by the holder for issuance such as blinded payloads, keys for replay prevention, etc.

All algorithms MUST provide integrity protection for the Issuer Header and all payloads and MUST specify all digest and/or hash2curve methods used.

5.2. Confirm

Performed by the holder to validate that the issued JWP is correctly formed and protected.

Each algorithm MAY support using additional input items options, such as those sent to the issuer for issuance. After confirmation, an algorithm MAY return a modified JWP for serialized storage without the local state (such as with blinded payloads now unblinded).

The algorithm MUST fully verify the issued proof value against the Issuer Header and all payloads. If given a presented JWP instead of an issued one, the confirm process MUST return an error.

5.3. Present

Used to apply any selective disclosure choices and perform any unlinkability transformations, as well as to show binding.

An algorithm MAY support additional input options from the requesting party, such as for predicate proofs and verifiable computation requests.

Every algorithm MUST support the ability to hide any or all payloads. It MUST always include the Issuer Header unmodified in the presentation.

The algorithm MUST replace the issued proof value and generate a new presented proof value. It also MUST include a new Presentation Header that provides replay protection.

5.4. Verify

Performed by the verifier to verify the Headers along with any disclosed payloads and/or assertions about them from the proving party, while also verifying they are the same payloads and ordering as witnessed by the issuer.

The algorithm MUST verify the integrity of all disclosed payloads and MUST also verify the integrity of both the Issuer and Presentation Headers.

If the presented proof contains any assertions about the hidden payloads, the algorithm MUST also verify all of those assertions. It MAY support additional options, such as those sent to the holder to generate the presentation.

If given an issued JWP for verification, the algorithm MUST return an error.

6. Algorithm Specifications

This section defines how to use specific algorithms for JWPs.

6.1. Single Use

The Single Use (SU) algorithm is based on composing multiple traditional asymmetric signatures into a single JWP proof. It enables a very simple form of selective disclosure without requiring any advanced cryptographic techniques.

It does not support unlinkability if the same JWP is presented multiple times, therefore when privacy is required the holder will need to interact with the issuer again to receive new single-use JWPs (dynamically or in batches).

6.1.1. JWS Algorithm

The Single Use algorithm uses multiple signing keys to protect the Header as well as individual payloads of an Issued JWP. The issuer uses a stable public key to sign each Header, and a per-JWP ephemeral key (conveyed within the Header) to protect the individual payloads. These signatures are all created using the same Asymmetric Algorithm, with the JOSE and COSE name/label of this algorithm being part of registration for a fully-specified Single Use algorithm identifier.

The Issuer Header also conveys a holder presentation key, an ephemeral asymmetric key meant to only be used for presenting a single JWP. The fully-specified algorithm the holder must use for presentations is also included. This algorithm MAY be different from the algorithm used by the issuer.

The chosen algorithms MUST be asymmetric signing algorithms, so that each signature can be verified without sharing any private values between the parties.

6.1.2. Holder Setup

In order to support the protection of a presentation by a holder to a verifier, the holder MUST use a Holder Presentation Key during the issuance and the presentation of every Single Use JWP. This Holder Presentation Key MUST be generated and used for only one JWP if unlinkability is desired.

The issuer MUST verify that the holder has possession of this key. The holder-issuer communication to exchange this information is out of scope of this specification, but can be accomplished by the holder using this key to generate a JWS that signs a value the issuer can verify as unique.

The issuer MUST determine an appropriate holder presentation algorithm corresponding to the holder presentation key. If the holder and verifier cannot be assumed to know this algorithm is the appropriate choice for a given holder presentation key, this value MUST be conveyed in the hpa Issuer Header.

6.1.3. Issuer Setup

To create a Single Use JWP, the issuer first generates a unique Ephemeral Key using the selected internal algorithm. This key-pair will be used to sign each of the payloads of a single JWP and then discarded.

6.1.4. Signing Payloads

Each individual payload is signed using the selected internal algorithm using the Ephemeral Key.

6.1.5. Issuer Header

The Issuer’s Ephemeral Key MUST be included via the Issuer Ephemeral Key Header Parameter.

The Holder's Presentation Key MUST be included via the Holder Presentation Key Header Parameter.

The Holder's Presentation Algorithm MUST be included via the Holder Presentation Algorithm Header Parameter unless there is another way for the holder and verifier to unambiguously determine the appropriate algorithm to use.

The Issuer Header is signed using the appropriate internal signing algorithm for the given fully-specified single use algorithm, using the issuer's Stable Key.

6.1.6. Payloads

Each JWP payload is processed in order and signed using the given JWA using the issuer's Ephemeral Key.

6.1.7. Proof

The proof value is an octet string array. The first entry is the octet string of the Issuer Header signature, with an additional entry for each payload signature.

6.1.8. Presentation Header #{presentation-protected-header}

To generate a new presentation, the holder first creates a Presentation Header that is specific to the verifier being presented to. This Header MUST contain a parameter that both the holder and verifier trust as being unique and non-replayable. Use of the nonce Header Parameter is RECOMMENDED for this purpose.

This specification registers the nonce Header Parameter for the Presentation Header that contains a string value either generated by the verifier or derived from values provided by the verifier. When present, the verifier MUST ensure the nonce value matches during verification.

The Presentation Header MAY contain other Header Parameters that are either provided by the verifier or by the holder. These Presentation Header Parameters SHOULD NOT contain values that are common across multiple presentations and SHOULD be unique to a single presentation and verifier.

The Presentation Header MUST contain the same Algorithm protected header as the Issuer Header. The Holder Presentation Algorithm Header Parameter MUST NOT be included.

6.1.9. Presentation

The holder derives a new proof as part of presentation. The holder will also use these components to generate a presentation internal representation (#presentation-internal-representation). The number of components depends on the number of payloads which are being disclosed in the presented JWP.

The first proof component will be the signature over the Issuer Header made by the issuer's Stable Key.

For each payload which is to be disclosed, the corresponding payload signature (from the issued JWP) is included as a subsequent proof component. If the payload is being omitted, the corresponding payload signature is omitted from the proof components.

The Presentation Header, Issuer Header, payload slots (distinguishing which are being disclosed) and these proof components are inputs to determine the presentation internal representation.

The holder's signature over the presentation internal representation (using the holder's private key and the holder presentation algorithm) is then included as one additional proof component in the final presentation.

For example, if only the second and fifth of five payloads are being disclosed, then the proof at this stage will consist of three values:

1. The issuer's signature over the Issuer Header
2. The payload signature corresponding to the second payload
3. The payload signature corresponding to the fifth payload.

The presentation internal representation would be calculated with these three proof components, while the final presentation would have an additional fourth component containing the signature using the holder's private key.

Since the individual signatures in the proof value are unique and remain unchanged across multiple presentations, a Single Use JWP SHOULD only be presented a single time to each verifier in order for the holder to remain unlinkable across multiple presentations.

6.1.10. Verification of Presentation

Verification is performed using the following steps.

1. Check that the number of proof components is appropriate for the number of disclosed payloads. There MUST be two more proof components than disclosed payloads.
2. Verify the first proof component is a valid signature over Issuer Header octets, using the issuer’s stable key.
3. Extract the holder presentation key and holder presentation algorithm (if present) from the Issuer Header.
4. Omitting the final payload component, calculate the presentation internal representation (#presentation-internal-representation).
5. Verify the final proof component is a valid signature over the presentation internal binary form, using the holder’s presentation key and the extracted (or otherwise determined) holder presentation algorithm.
6. For each remaining proof component, verify they form a valid signature over each disclosed payload in sequence, using the issuer’s ephemeral key.

6.1.11. JPA Registration

The proposed JWP alg value is of the format "SU-" appended with the relevant JWS alg value for the chosen public and ephemeral key-pair algorithm, for example "SU-ES256".

6.2. Presentation Internal Representation

Some algorithms (such as Single use and MAC) use a holder key to provide integrity over the presentation. For these algorithms, an internal binary form of the presentation must be generated both for signing by the holder, and for verification by the verifier. Other algorithms MAY use this same form for consistency.

The instructions for creating this binary representation will also create well-formed CBOR, although this data is not meant to be shared outside the implementing algorithm. Instead, it focuses on simplicity of generation by the holder and verifier implementations. Although CBOR has multiple representations of the same underlying information, this same octet string MUST be generated by an implementation.

When a length or count is added by the steps below, it is added as its 8-byte, network-ordered representation. For example, the length of a 1,234 byte payload would have a length representation of 0x00 00 00 00 00 00 04 D2.

The binary representation is created by appending data into a single octet string in the following order:

1. 0x84 5B

2. The length and octets of the Presentation Header

3. 0x5B

4. The length and octets of the Issuer Header

5. 0x9B

6. The number of payload slots in the issued message

7. For each payload representation:

   * If the payload is being omitted, the value 0xF6

   * Otherwise:

     1. 0x5B
     2. The length and octets of the payload

8. 0x9B

9. The number of proof components as specified by the algorithm

10. For each proof component, append:

    1. 0x5B
    2. The length and octets of the proof component

6.3. BBS

The BBS Signature Scheme [I-D.irtf-cfrg-bbs-signatures] is under active development within the CRFG.

This algorithm supports both selective disclosure and unlinkability, enabling the holder to generate multiple presentations from one issued JWP without a verifier being able to correlate those presentations together based on the proof.

6.3.1. JPA Algorithms

The BBS algorithm corresponds to a cipher suite identifier of BBS_BLS12381G1_XMD:SHA-256_SSWU_RO_.

6.3.2. Key Format

The key used for the BBS algorithm is an elliptic curve-based key pair, specifically against the G_2 subgroup of a pairing friendly curve. Additional details on key generation can be found in Section 3.4. The JWK and Cose Key Object representations of the key are detailed in [I-D.ietf-cose-bls-key-representations].

There is no additional holder presentation key necessary for presentation proofs.

6.3.3. Issuance

Issuance is performed using the Sign operation from Section 3.5.1 of [I-D.irtf-cfrg-bbs-signatures]. This operation utilizes the issuer's BLS12-381 G2 key pair as SK and PK, along with desired Header octets as header, and the array of payload octet string as messages.

The octets resulting from this operation form a single octet string in the issuance proof array, to be used along with the Header and payloads to serialize the JWP.

6.3.4. Issuance Proof Verification

Holder verification of the signature on issuance form is performed using the Verify operation from [@!I-D.irtf-cfrg-bbs-signatures, section 3.5.2].

This operation utilizes the issuer's public key as PK, the proof as signature, the Header octets as header and the array of payload octets as messages.

6.3.5. Presentation

Derivation of a presentation is done by the holder using the ProofGen operation from Section 3.5.3 of [I-D.irtf-cfrg-bbs-signatures].

This operation utilizes the issuer's public key as PK, the Issuer Header as header, the issuance proof as signature, the issuance payloads as messages, and the holder's Presentation Header as ph.

The operation also takes a vector of indexes into messages, describing which payloads the holder wishes to disclose. All payloads are required for proof generation, but only these indicated payloads will be required to be disclosed for later proof verification.

The output of this operation is the presentation proof, as a single octet string.

Presentation serialization leverages the two Headers and presentation proof, along with the disclosed payloads. Non-disclosed payloads are represented with the absent value of null in CBOR serialization and a zero-length string in compact serialization.

6.3.6. Presentation Verification

Verification of a presentation is done by the verifier using the ProofVerify operation from [@!I-D.irtf-cfrg-bbs-signatures, Section 3.5.4].

This operation utilizes the issuer’s public key as PK, the Issuer Header as header, the issuance proof as signature, the holder’s Presentation Header as ph, and the payloads as disclosed_messages.

In addition, the disclosed_indexes scalar array is calculated from the payloads provided. Values disclosed in the presented payloads have a zero-based index in this array, while the indices of absent payloads are omitted.

6.4. Message Authentication Code

The Message Authentication Code (MAC) JPA uses a MAC to both generate ephemeral secrets and to authenticate payloads, along with an asymmetric signature to provide integrity to the issued JWP.

The holder can manipulate which payloads are disclosed from the issued JWP, and uses the Holder Presentation Key to create a presentation. The signature created from the Holder Presentation Key MAY use a different algorithm than the Issuer used to sign the issued form.

Like the Single Use algorithm family, it also does not support unlinkability if the same JWP is presented multiple times and requires an individually issued JWP for each presentation in order to fully protect privacy. When compared to the JWS approach, using a MAC requires less computation but can result in potentially larger presentation proof values.

The design is intentionally minimal and only involves using a single standardized MAC method instead of a mix of MAC/hash methods or a custom hash-based construct. It is able to use any published cryptographic MAC method such as HMAC [RFC2104] or KMAC (https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf). It uses traditional public key-based signatures to verify the authenticity of the issuer and holder.

6.4.1. Holder Setup

In order to support the protection of a presentation by a holder to a verifier, the holder MUST use a Holder Presentation Key during the issuance and the presentation of every MAC JWP. This Holder Presentation Key MUST be generated and used for only one JWP if unlinkability is desired.

The issuer MUST verify that the holder has possession of this key. The holder-issuer communication to exchange this information is out of scope of this specification, but can be accomplished by the holder using this key to generate a JWS that signs a value the issuer can verify as unique.

The holder’s presentation key MUST be included in the Issuer Header using the Holder Presentation Key Header Parameter.

The issuer MUST determine an appropriate holder presentation algorithm corresponding to the holder presentation key. If the holder and verifier cannot be assumed to know this algorithm is the appropriate choice for a given holder presentation key, this value MUST be conveyed in the Holder Protected Algorithm Header Parameter.

6.4.2. Issuer Setup

To use the MAC algorithm, the issuer must have a stable public key pair to perform signing. To start the issuance process, a single 32-byte random Shared Secret must first be generated. This value will be shared privately with the holder as part of the issuer’s JWP proof value.

The Shared Secret is used by both the issuer and holder as the MAC method’s key to generate a new set of unique ephemeral keys. These keys are then used as the input to generate a MAC that protects each payload.

6.4.3. Combined MAC Representation

The combined MAC representation is a single octet string representing the MAC values of the Issuer Header, along with each payload provided by the issuer. This representation is signed by the issuer, but not shared - parties will recreate this octet string and verify the signature to verify the integrity of supplied Issuer Header and the integrity of any disclosed payloads.

The steps below describe a sequential concatenation of binary values to generate the Combined MAC Representation. The instructions for generating this octet string will also generate well-formed CBOR, although this data is not meant to be shared outside the implementing algorithm. Instead, it focuses on simplicity of generation by the issuer, holder, and verifier implementations. Although CBOR has multiple representations of the same underlying information, this same octet string MUST be generated by an implementation.

When a length or count is added by steps in this section, it is added as its 8-byte, network-ordered representation. For example, the length of a 1,234-byte payload would have a length representation of 0x00 00 00 00 00 00 04 D2.

The holder will a unique key per payload value using a MAC, with the Shared Secret as the key and a generated binary value. This binary value is constructed by appending data into a single octet string:

1. 0x82 67 70 61 79 6C 6F 61 64 1B
2. The zero indexed count of the payload slot

The holder will also compute a corresponding MAC of each payload. This MAC uses the unique key above and the payload octet string as the value.

When verifying a presentation, the shared secret will be unavailable so the unique key cannot be calculated. The payload octet string may also be omitted in the presentation. The following instructions describe how to get the corresponding MAC of each payload:

* If the payload is disclosed, the corresponding proof component (as described in MAC Presentation Proof (#mac-presentation-proof)) will contain the generated unique key. The payload MAC will be calculated using this key and the payload octets as the value.
* If the payload is not disclosed, the corresponding proof component will be the payload MAC.

The binary representation is created by appending data into a single octet string in the following order:

1. 0x82 5B
2. The length and octets of the Issuer Header
3. 0x9B
4. The number of payload slots in the issued JWP
5. For each payload representation:
    1. 0x5B
    2. The length and value of the per payload MAC

6.4.4. Issuer Header

The Holder’s Presentation Key MUST be included via the Holder Presentation Key Header Parameter.

The Holder’s Presentation Algorithm MUST be included via the Holder Presentation Algorithm Header Parameter unless there is another way for the holder and verifier to unambiguously determine the appropriate algorithm to use.

6.4.5. Issuer Proof

The issuer proof consists of two octet strings.

The first octet string is the issuer signature over the combined MAC representation. The issuer signs the combined MAC representation using its stable public key, and the internal signing algorithm for the given fully-specified MAC algorithm variant.

The second octet string is the Shared Secret used to generate the per-payload keys for the combined representation.

6.4.6. Presentation Header

See the Presentation Header (#presentation-protected-header) section given for Single Use algorithms.

6.4.7. Presentation Proof

The presentation proof is made of multiple components.

The first proof component is the issuer signature over the Combined MAC Representation, which is provided as the first proof component from the issued form.

There will now be one proof component per payload slot in the issued JWP. These are used by the verifier to reconstruct the combined MAC representation without access to the Shared Secret. The proof components are calculated per the instructions used to generate the Combined MAC Representation (#combined-mac-representation)

If a payload is disclosed, the corresponding proof component will be the unique key.

If a payload is not disclosed, the corresponding proof component will be the payload's MAC (using the unique key.)

The Presentation Header, Issuer Header, payload slots (distinguishing which are being disclosed) and above proof components are inputs to determine the presentation internal representation (#presentation-internal-representation).

The holder's signature over the presentation internal representation (using the holder's private key and the holder presentation algorithm) is then included as one additional proof component in the final presentation.

The presented form should have two more proof components than payload slots in the issued JWP.

Note that the second component of the issued JWP is a shared secret for use by the holder to generate the unique keys used in the Combined MAC Representation. This MUST NOT be included in the presentation.

6.4.8. Verification of the Presentation Proof

Verification is performed using the following steps.

1. Check the number of proof components is appropriate for the number of disclosed payloads. There MUST be two more proof components than disclosed payloads.
2. Using the fully-specified MAC algorithm in use, use the Issuer Header, disclosed payloads, and the proof components corresponding to the payloads to regenerate the Combined MAC Representation.
3. Verify the first proof component is a valid signature over the Issuer Header octets, using the issuer's stable key.
4. Extract the holder presentation key and holder presentation algorithm (if present) from the Issuer Header.
5. Omitting the final payload component, calculate the presentation internal representation (#presentation-internal-representation).
6. Verify the final proof component is a valid signature over the presentation internal binary form, using the holder's presentation key and the extracted (or otherwise determined) holder presentation algorithm.

6.4.9. JPA Registration

Proposed JWP alg value is of the format "MAC-" appended with a unique identifier for the set of MAC and signing algorithms used. Below are the initial registrations:

* MAC-H256 uses HMAC SHA-256 as the MAC and ECDSA using P-256 and SHA-256 for the signatures
* MAC-H384 uses HMAC SHA-384 as the MAC and ECDSA using P-384 and SHA-384 for the signatures
* MAC-H512 uses HMAC SHA-512 as the MAC and ECDSA using P-521 and SHA-512 for the signatures
* MAC-K25519 uses KMAC SHAKE128 as the MAC and EdDSA using Curve25519 for the signatures
* MAC-K448 uses KMAC SHAKE256 as the MAC and EdDSA using Curve448 for the signatures
* MAC-H256K uses HMAC SHA-256 as the MAC and ECDSA using secp256k1 and SHA-256 for the signatures

7. Security Considerations

> Editor’s Note: This will follow once the algorithms defined here have become more stable.

* Data minimization of the proof value
* Unlinkability of the Header contents

8. IANA Considerations

The following registration procedure is used for all the registries established by this specification.

Values are registered on a Specification Required [RFC5226] basis after a three-week review period on the jose-reg-review@ietf.org (mailto:jose-reg-review@ietf.org) mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Experts may approve registration once they are satisfied that such a specification will be published.

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register JWP algorithm: example").

Within the review period, the Designated Experts will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG's attention (using the iesg@ietf.org (mailto:iesg@ietf.org) mailing list) for resolution.

Criteria that should be applied by the Designated Experts include determining whether the proposed registration duplicates existing functionality, whether it is likely to be of general applicability or useful only for a single application, and whether the registration description is clear.

IANA must only accept registry updates from the Designated Experts and should direct all requests for registration to the review mailing list.

It is suggested that multiple Designated Experts be appointed who are able to represent the perspectives of different applications using this specification, in order to enable broadly informed review of registration decisions. In cases where a registration decision could be perceived as creating a conflict of interest for a particular Expert, that Expert should defer to the judgment of the other Experts.

8.1. JSON Web Proof Algorithms Registry

This specification establishes the IANA "JSON Web Proof Algorithms" registry, under the "JSON Object Signing and Encryption (JOSE)" registry group. The registry records values values of the JWP alg (algorithm) Header Parameter. The registry records the algorithm name, the algorithm description, the algorithm usage locations, the implementation requirements, the change controller, and a reference to the specification that defines it. The same algorithm name can be registered multiple times, provided that the sets of usage locations are disjoint.

It is suggested that the length of the key be included in the algorithm name when multiple variations of algorithms are being registered that use keys of different lengths and the key lengths for each need to be fixed (for instance, because they will be created by key derivation functions). This allows readers of the JSON text to more easily make security decisions.

The Designated Experts should perform reasonable due diligence that algorithms being registered either are currently considered cryptographically credible or are being registered as Deprecated or Prohibited.

The implementation requirements of an algorithm may be changed over time as the cryptographic landscape evolves, for instance, to change the status of an algorithm to Deprecated or to change the status of an algorithm from Optional to Recommended+ or Required. Changes of implementation requirements are only permitted on a Specification Required basis after review by the Designated Experts, with the new specification defining the revised implementation requirements level.

8.1.1. Registration Template

Algorithm Name: Brief descriptive name of the algorithm (e.g., Single-Use JWP using ES256.) Descriptive names may not match other registered names unless the Designated Experts state that there is a compelling reason to allow an exception.

Algorithm JSON Label: The string label requested (e.g., SU-ES256). This label is a case-sensitive ASCII string. JSON Labels may not match other registered labels in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Algorithm CBOR Label: The integer label requested (e.g., 1). CBOR Labels may not match other registered labels unless the Designated Experts state that there is a compelling reason to allow an exception.

Algorithm Description: Optional additional information clarifying the algorithm. This may be used for example to document additional chosen parameters.

Algorithm Usage Location(s): The algorithm usage locations, which should be one or more of the values Issued or Presented. Other values may be used with the approval of a Designated Expert.

JWP Implementation Requirements: The algorithm implementation requirements for JWP, which must be one of the words Required, Recommended, Optional, Deprecated, or Prohibited. Optionally, the word can be followed by a + or -. The use of + indicates that the requirement strength is likely to be increased in a future version of the specification. The use of - indicates that the requirement strength is likely to be decreased in a future version of the specification. Any identifiers registered for algorithms that are otherwise unsuitable for direct use as JWP algorithms must be registered as Prohibited.

Change Controller: For IETF Stream RFCs, list the IETF. For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s): Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

Algorithm Analysis Documents(s): References to a publication or publications in well-known cryptographic conferences, by national standards bodies, or by other authoritative sources analyzing the cryptographic soundness of the algorithm to be registered. The Designated Experts may require convincing evidence of the cryptographic soundness of a new algorithm to be provided with the registration request unless the algorithm is being registered as Deprecated or Prohibited. Having gone through working group and IETF review, the initial registrations made by this document are exempt from the need to provide this information.

8.1.2. Initial Registry Contents

8.1.2.1. Single-Use JWP using ES256 Algorithm

* Algorithm Name: Single-Use JWP using ES256
* Algorithm JSON Label: SU-ES256
* Algorithm CBOR Label: 1
* Algorithm Usage Location(s): Issued, Presented
* JWP Implementation Requirements: Recommended
* Change Controller: IETF
* Specification Document(s): Section 6.1.11 of this specification
* Algorithm Analysis Documents(s): n/a

8.1.2.2. Single-Use JWP using ES384 Algorithm

* Algorithm Name: Single-Use JWP using ES384
* Algorithm JSON Label: SU-ES384
* Algorithm CBOR Label: 2
* Algorithm Usage Location(s): Issued, Presented
* JWP Implementation Requirements: Optional
* Change Controller: IETF
* Specification Document(s): Section 6.1.11 of this specification
* Algorithm Analysis Documents(s): n/a

8.1.2.3. Single-Use JWP using ES512 Algorithm

* Algorithm Name: Single-Use JWP using ES512
* Algorithm JSON Label: SU-ES512
* Algorithm CBOR Label: 3
* Algorithm Usage Location(s): Issued, Presented
* JWP Implementation Requirements: Optional
* Change Controller: IETF
* Specification Document(s): Section 6.1.11 of this specification
* Algorithm Analysis Documents(s): n/a

8.1.2.4. BBS using SHA-256 Algorithm

* Algorithm Name: BBS using SHA-256

* Algorithm JSON Label: BBS
* Algorithm CBOR Label: 4
* Algorithm Description: Corresponds to a cipher suite identifier of BBS_BLS12381G1_XMD:SHA-256_SSWU_RO_H2G_HM2S_
* Algorithm Usage Location(s): Issued, Presented
* JWP Implementation Requirements: Required
* Change Controller: IETF
* Specification Document(s): Section 6.3.1 of this specification
* Algorithm Analysis Documents(s): n/a

8.1.2.5. MAC-H256 Algorithm

* Algorithm Name: MAC-H256
* Algorithm JSON Label: MAC-H256
* Algorithm CBOR Label: 5
* Algorithm Description: MAC-H256 uses HMAC SHA-256 as the MAC, and ECDSA using P-256 and SHA-256 for the signatures
* Algorithm Usage Location(s): Issued, Presented
* JWP Implementation Requirements: Optional
* Change Controller: IETF
* Specification Document(s): Section 6.4.9 of this specification
* Algorithm Analysis Documents(s): n/a

8.1.2.6. MAC-H384 Algorithm

* Algorithm Name: MAC-H384
* Algorithm JSON Label: MAC-H384
* Algorithm CBOR Label: 6
* Algorithm Description: MAC-H384 uses HMAC SHA-384 as the MAC, and ECDSA using P-384 and SHA-384 for the signatures
* Algorithm Usage Location(s): Issued, Presented
* JWP Implementation Requirements: Optional
* Change Controller: IETF
* Specification Document(s): Section 6.4.9 of this specification
* Algorithm Analysis Documents(s): n/a

8.1.2.7. MAC-H512 Algorithm

* Algorithm Name: MAC-H512
* Algorithm JSON Label: MAC-H512
* Algorithm CBOR Label: 7
* Algorithm Description: MAC-H512 uses HMAC SHA-512 as the MAC, and ECDSA using P-521 and SHA-512 for the signatures
* Algorithm Usage Location(s): Issued, Presented
* JWP Implementation Requirements: Optional
* Change Controller: IETF
* Specification Document(s): Section 6.4.9 of this specification
* Algorithm Analysis Documents(s): n/a

8.1.2.8. MAC-K25519 Algorithm

* Algorithm Name: MAC-K25519
* Algorithm JSON Label: MAC-K25519
* Algorithm CBOR Label: 8
* Algorithm Description: MAC-K25519 uses KMAC SHAKE128 as the MAC, and EdDSA using Curve25519 for the signatures
* Algorithm Usage Location(s): Issued, Presented
* JWP Implementation Requirements: Optional
* Change Controller: IETF
* Specification Document(s): Section 6.4.9 of this specification
* Algorithm Analysis Documents(s): n/a

8.1.2.9. MAC-K448 Algorithm

* Algorithm Name: MAC-K448
* Algorithm JSON Label: MAC-K448
* Algorithm CBOR Label: 9
* Algorithm Description: MAC-K448 uses KMAC SHAKE256 as the MAC, and EdDSA using Curve448 for the signatures
* Algorithm Usage Location(s): Issued, Presented
* JWP Implementation Requirements: Optional
* Change Controller: IETF
* Specification Document(s): Section 6.4.9 of this specification
* Algorithm Analysis Documents(s): n/a

8.1.2.10. MAC-H256K Algorithm

* Algorithm Name: MAC-H256K
* Algorithm JSON Label: MAC-H256K
* Algorithm CBOR Label: 10
* Algorithm Description: MAC-H256K uses HMAC SHA-256 as the MAC, and ECDSA using secp256k1 and SHA-256 for the signatures
* Algorithm Usage Location(s): Issued, Presented
* JWP Implementation Requirements: Optional
* Change Controller: IETF
* Specification Document(s): Section 6.4.9 of this specification
* Algorithm Analysis Documents(s): n/a

9. References

9.1. Normative References

[I-D.ietf-jose-json-web-proof]
Waite, D., Jones, M. B., and J. Miller, "JSON Web Proof", Work in Progress, Internet-Draft, draft-ietf-jose-json-web-proof-latest, <https://datatracker.ietf.org/doc/html/draft-ietf-jose-json-web-proof>.

[I-D.irtf-cfrg-bbs-signatures]
Looker, T., Kalos, V., Whitehead, A., and M. Lodder, "The BBS Signature Scheme", Work in Progress, Internet-Draft, draft-irtf-cfrg-bbs-signatures-09, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bbs-signatures-09>.

[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>.

[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>.

[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>.

9.2. Informative References

[I-D.ietf-cbor-edn-literals]
Bormann, C., "CBOR Extended Diagnostic Notation (EDN)", Work in Progress, Internet-Draft, draft-ietf-cbor-edn-literals-19, 16 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cbor-edn-literals-19>.

[I-D.ietf-cose-bls-key-representations]
Looker, T. and M. B. Jones, "Barreto-Lynn-Scott Elliptic Curve Key Representations for JOSE and COSE", Work in Progress, Internet-Draft, draft-ietf-cose-bls-key-representations-08, 4 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-bls-key-representations-08>.

[I-D.ietf-spice-oidc-cwt]
Maldant, B. and M. B. Jones, "OpenID Connect Standard Claims Registration for CBOR Web Tokens", Work in Progress, Internet-Draft, draft-ietf-spice-oidc-cwt-02, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-spice-oidc-cwt-02>.

[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/info/rfc2104>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

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

[VC-DATA-MODEL-2.0] Sporny, M., Jr, T. T., Herman, I., Cohen, G., and M. B. Jones, "Verifiable Credentials Data Model v2.0", 15 May 2025, <https://www.w3.org/TR/vc-data-model-2.0>.

Appendix A. Example JWPs

The following examples use algorithms defined in JSON Proof Algorithms and also contain the keys used, so that implementations can validate these samples.

A.1. Example JSON-Serialized Single-Use JWP

This example uses the Single-Use Algorithm as defined in JSON Proof Algorithms to create a JSON Proof Token. It demonstrates how to apply selective disclosure using an array of traditional JWS-based signatures. Unlinkability is only achieved by using each JWP one time, as multiple uses are inherently linkable via the traditional ECDSA signature embedded in the proof.

To begin, we need two asymmetric keys for Single Use: one that represents the JPT Issuer’s stable key and the other is an ephemeral key generated by the Issuer just for this JWP.

This is the Issuer’s stable private key used in this example in the JWK format:


```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "MQGKGfEcKSXsHBhBSrDjYxvA6NqBh4X5ylt9rGPVuLg",
  "y": "JVguv8zBJUEsNqMAAlOqX_DigKPY4P3oR-LFCqOAvQ0",
  "d": "wilgoSc8vpMoxz8tJ4roRF9di9l9L4RukWrPT6wdZi8"
}
```


Figure 1: Issuer Private Key (ES256 in JWK)

This is the ephemeral private key used in this example in the JWK format:

```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "YrcAuI5hLgCRRfIMKeJ9276ld1sJ7oD8l-ctY4zfdEs",
  "y": "JBeTWoj1o2Dbwfw4oXbf1aQcuUzvNhRZbgGoh2uh7-M",
  "d": "0Vhqf4UN-2FQ4uFCxXGuza-8uX7ykM_S0CyUEkD5mDM"
}
```


Figure 2: Issuer Ephemeral Private Key (ES256 in JWK)

This is the Holder's presentation private key used in this example in the JWK format:


```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "9f65fOnwQnkAz9Xh0hrDqt_Zs4Fsu-b9woMh0V27JRo",
  "y": "x4Cg8h1q738lrnguGv5XkadWT59tfFytlMO3hVJMe_Q",
  "d": "T0hKcZ1iWLGrRSRzC9WZM3xAso9djnpYVNVeu3nql6A"
}
```


Figure 3: Holder Presentation Private Key (ES256 in JWK)

The Header declares that the data structure is a JPT and the JWP Proof Input is secured using the Single-Use ECDSA algorithm with the P-256 curve and SHA-256 digest. It also includes the ephemeral public key, the Holder's presentation public key and list of claims used for this JPT.

```
{
  "alg": "SU-ES256",
  "typ": "JPT",
  "iss": "https://issuer.example",
  "hpa": "ES256",
  "claims": [
    "iat",
    "exp",
    "family_name",
    "given_name",
    "email",
    "address",
    "age_over_21"
  ],
  "iek": {
    "kty": "EC",
    "crv": "P-256",
    "x": "YrcAuI5hLgCRRfIMKeJ9276ld1sJ7oD8l-ctY4zfdEs",
    "y": "JBeTWoj1o2Dbwfw4oXbf1aQcuUzvNhRZbgGoh2uh7-M"
  },
  "hpk": {
    "kty": "EC",
    "crv": "P-256",
    "x": "9f65fOnwQnkAz9Xh0hrDqt_Zs4Fsu-b9woMh0V27JRo",
    "y": "x4Cg8h1q738lrnguGv5XkadWT59tfFytlMO3hVJMe_Q"
  }
}
```


Figure 4: Issuer Header (SU-ES256, JSON)


```
eyJhbGciOiJTVS1FUzI1NiIsInR5cCI6IkpQVCIsImlzcyI6Imh0dHBzOi8vaXNzdWVyL
mV4YW1wbGUiLCJocGEiOiJFUzI1NiIsImNsYWltcyI6WyJpYXQiLCJleHAiLCJmYW1pbH
lfbmFtZSIsImdpdmVuX25hbWUiLCJlbWFpbCIsImFkZHJlc3MiLCJhZ2Vfb3Zlcl8yMSJ
dLCJpZWsiOnsia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIsIngiOiJZcmNBdUk1aExnQ1JS
ZklNS2VKOTI3NmxkMXNKN29EOGwtY3RZNHpmZEVzIiwieSI6IkpCZVRXb2oxbzJEYndmd
zRvWGJmMWFRY3VVenZOaFJaYmdHb2gydWg3LU0ifSwiaHBrIjp7Imt0eSI6IkVDIiwiY3
J2IjoiUC0yNTYiLCJ4IjoiOWY2NWZPbndRbmtBejlYaDBockRxdF9aczRGc3UtYjl3b01
oMFYyN0pSbyIsInkiOiJ4NENnOGgxcTczOGxybmd1R3Y1WGthZFdUNTl0ZkZ5dGxNTzNo
VkpNZV9RIn19
```


Figure 5: Encoded Issuer Header (SU-ES256, JSON, encoded)

The Single Use algorithm utilizes multiple individual JWS Signatures. Each signature value is generated by creating a JWS with a single Header with the associated alg value. In this example, the fixed Header used for each JWS is the serialized JSON Object {"alg":"ES256"}. This Header will be used to generate a signature over each corresponding payload in the JWP. The corresponding octet value in the proof is the octet string (base64url-decoded) value of the signature.

The final proof value from the Issuer is an array with the octets of the Header signature, followed by entries for each payload signature.


```
[
    1714521600,
    1717199999,
    "Doe",
    "Jay",
    "jaydoe@example.org",
    {
        "formatted": "1234 Main St.\nAnytown, CA 12345\nUSA",
        "street_address": "1234 Main St.",
        "locality": "Anytown",
        "region": "CA",
        "postal_code": 12345,
        "country": "USA"
    },
    true
]
```


Figure 6: Issuer payloads (JSON, as array)

The compact serialization of the same JPT is:

```
eyJhbGciOiJTVS1FUzI1NiIsInR5cCI6IkpQVCIsImlzcyI6Imh0dHBzOi8vaXNzdWVyL
mV4YW1wbGUiLCJocGEiOiJFUzI1NiIsImNsYWltcyI6WyJpYXQiLCJleHAiLCJmYW1pbH
lfbmFtZSIsImdpdmVuX25hbWUiLCJlbWFpbCIsImFkZHJlc3MiLCJhZ2Vfb3Zlcl8yMSJ
dLCJpZWsiOnsia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIsIngiOiJZcmNBdUk1aExnQ1JS
ZklNS2VKOTI3NmxkMXNKN29EOGwtY3RZNHpmZEVzIiwieSI6IkpCZVRXb2oxbzJEYndmd
zRvWGJmMWFRY3VVenZOaFJaYmdHb2gydWg3LU0ifSwiaHBrIjp7Imt0eSI6IkVDIiwiY3
J2IjoiUC0yNTYiLCJ4IjoiOWY2NWZPbndRbmtBejlYaDBockRxdF9aczRGc3UtYjl3b01
oMFYyN0pSbyIsInkiOiJ4NENnOGgxcTczOGxybmd1R3Y1WGthZFdUNTl0ZkZ5dGxNTzNo
VkpNZV9RIn19.MTcxNDUyMTYwMA~MTcxNzE5OTk5OQ~IkRvZSI~IkpheSI~ImpheWRvZU
BleGFtcGxlLm9yZyI~eyJmb3JtYXR0ZWQiOiIxMjM0IE1haW4gU3QuXG5Bbnl0b3duLCB
DQSAxMjM0NVxuVVNBIiwic3RyZWV0X2FkZHJlc3MiOiIxMjM0IE1haW4gU3QuIiwibG9j
YWxpdHkiOiJBbnl0b3duIiwicmVnaW9uIjoiQ0EiLCJwb3N0YWxfY29kZSI6MTIzNDUsI
mNvdW50cnkiOiJVU0EifQ~dHJ1ZQ.4W_OybS-VeHlWQLfWJxh2SlA2PWCZLqR7YayOUbD
QkBtfxDCTaJjqGOr7oJsDNYAZQDOTiLHJv-OFc7ScpAV2Q~qrPb-ak5Qt-oc2Ucz-WX0D
4MrYDRxPQPIRCZiSe8xriInZYauyQudWAuxXM4LotSTlyD-l8zingfvEA5ad9s6A~irJA
Qg_EBwdkAGrpcpl8xgybJoVG1CI-UF46_0WkcjwL8rAGkdOu98-iQ6ns5ilgKpH2qPlW_
fEcwpm9FoBLEg~CzshgACXtFzuixPAXDTNmUyl-pw3NgM4xHeF32A3kLT1PWQLmTthFrI
NBfVKcN7XisFe5dSTjKO8_cNdI6CG8Q~_RnyKDe5SZg4raS9TUfLyWOrD6HF_hAmq35Zu
gQlRxCOYbk9mKqj7onE5017rMI7xFKymJa9MKcI6_udGUUHSQ~XJ3f2Rs46ihHlbb2uRQ
haFrdDiLvd21EnC6LN_KLjMgyIZ-IqunRHAsjwz-zjaSLKz2QJTwqZlVa0MF_47VorQ~J
Iqu0pEZlrkO2F3XXuGU5FgJPdoORNlM9RDhp5miVjPVfQt2f4u_VYdgyOivVJlMzZuOd5
G59mhIOEiWKy6t-A~cWPuj_C33X2Zu9xADB1Yu6TpC008OXNP0FHfRcNY1xBb1kxHmOXj
qNh0Ei4h9VVeEuuuwir7_VDYMM_KhvUpkw
```


Figure 7: Issued JWP (SU-ES256, JSON, Compact Serialization)

To present this JPT, we first use the following Presentation Header with a nonce (provided by the Verifier):


```
{
  "alg": "SU-ES256",
  "aud": "https://recipient.example.com",
  "nonce": "38c7uOSi9raTnAmXDaFaoWEj9_JQL2nOfhqH7YiCwMo"
}
```


Figure 8: Presentation Header (SU-ES256, JSON)


```
eyJhbGciOiJTVS1FUzI1NiIsImF1ZCI6Imh0dHBzOi8vcmVjaXBpZW50LmV4YW1wbGUuY
29tIiwibm9uY2UiOiIzOGM3dU9TaTlyYVRuQW1YRGFGYW9XRWo5X0pRTDJuT2ZocUg3WW
lDd01vIn0
```


Figure 9: Presentation Header (SU-ES256, JSON, Base64url-Encoded)

We apply selective disclosure of only the given name and age claims (family name and email hidden), and remove the proof components corresponding to these entries.

Using the selectively disclosed information, we generate the presentation internal representation. Using that and the selectively disclosed payloads, we get the following presented JPT in compact serialization:


```
eyJhbGciOiJTVS1FUzI1NiIsImF1ZCI6Imh0dHBzOi8vcmVjaXBpZW50LmV4YW1wbGUuY
29tIiwibm9uY2UiOiIzOGM3dU9TaTlyYVRuQW1YRGFGYW9XRWo5X0pRTDJuT2ZocUg3WW
lDd01vIn0.eyJhbGciOiJTVS1FUzI1NiIsInR5cCI6IkpQVCIsImlzcyI6Imh0dHBzOi8
vaXNzdWVyLmV4YW1wbGUiLCJocGEiOiJFUzI1NiIsImNsYWltcyI6WyJpYXQiLCJleHAi
LCJmYW1pbHlfbmFtZSIsImdpdmVuX25hbWUiLCJlbWFpbCIsImFkZHJlc3MiLCJhZ2Vfb
3Zlcl8yMSJdLCJpZWsiOnsia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIsIngiOiJZcmNBdU
k1aExnQ1JSZklNS2VKOTI3NmxkMXNKN29EOGwtY3RZNHpmZEVzIiwieSI6IkpCZVRXb2o
xbzJEYndmdzRvWGJmMWFRY3VVenZOaFJaYmdHb2gydWg3LU0ifSwiaHBrIjp7Imt0eSI6
IkVDIiwiY3J2IjoiUC0yNTYiLCJ4IjoiOWY2NWZPbndRbmtBejlYaDBockRxdF9aczRGc
3UtYjl3b01oMFYyN0pSbyIsInkiOiJ4NENnOGgxcTczOGxybmd1R3Y1WGthZFdUNTl0Zk
Z5dGxNTzNoVkpNZV9RIn19.MTcxNDUyMTYwMA~MTcxNzE5OTk5OQ~IkRvZSI~IkpheSI~
ImpheWRvZUBleGFtcGxlLm9yZyI~eyJmb3JtYXR0ZWQiOiIxMjM0IE1haW4gU3QuXG5Bb
nl0b3duLCBDQSAxMjM0NVxuVVNBIiwic3RyZWV0X2FkZHJlc3MiOiIxMjM0IE1haW4gU3
QuIiwibG9jYWxpdHkiOiJBbnl0b3duIiwicmVnaW9uIjoiQ0EiLCJwb3N0YWxfY29kZSI
6MTIzNDUsImNvdW50cnkiOiJVU0EifQ~dHJ1ZQ~~.4W_OybS-VeHlWQLfWJxh2SlA2PWC
ZLqR7YayOUbDQkBtfxDCTaJjqGOr7oJsDNYAZQDOTiLHJv-OFc7ScpAV2Q~qrPb-ak5Qt
-oc2Ucz-WX0D4MrYDRxPQPIRCZiSe8xriInZYauyQudWAuxXM4LotSTlyD-l8zingfvEA
5ad9s6A~irJAQg_EBwdkAGrpcpl8xgybJoVG1CI-UF46_0WkcjwL8rAGkdOu98-iQ6ns5
ilgKpH2qPlW_fEcwpm9FoBLEg~CzshgACXtFzuixPAXDTNmUyl-pw3NgM4xHeF32A3kLT
1PWQLmTthFrINBfVKcN7XisFe5dSTjKO8_cNdI6CG8Q~_RnyKDe5SZg4raS9TUfLyWOrD
6HF_hAmq35ZugQlRxCOYbk9mKqj7onE5017rMI7xFKymJa9MKcI6_udGUUHSQ~XJ3f2Rs
46ihHlbb2uRQhaFrdDiLvd21EnC6LN_KLjMgyIZ-IqunRHAsjwz-zjaSLKz2QJTwqZlVa
0MF_47VorQ~hRXfnQlPqGHS_l34OqqonK59sgfv-vVnEZ4GD1ePkflRK5Djvj7FCZ3nbx
56sk4sIptuoxB7w5RDvUMrA5T78w
```


| Figure: Presentation (SU-ES256, JSON, Compact Serialization)

A.2. Example CBOR-Serialized Single-Use CPT

This example is meant to mirror the prior compact serialization, using RFC8392 (CWT) and claims from [I-D.ietf-spice-oidc-cwt], illustrated using [I-D.ietf-cbor-edn-literals] (EDN).

To simplify this example, the same information is represented as the JPT example above, including the same public and private keys.

```
{          / protected header /
  1: 1,     / alg: "SU-ES256" /
  3: 20,    / typ: "JPT" (20CPA) /
  5: "https://issuer.example",  / iss: "https://issuer.example" /
  6: [      / claims /
    6,      / "iat" /
    4,      / "exp" /
    170,    / "family_name" (I-D.maldant-spice-oidc-cwt TBD1) /
    171,    / "given_name"  (I-D.maldant-spice-oidc-cwt TBD2) /
    179,    / "email"       (I-D.maldant-spice-oidc-cwt TBD10) /
    187,    / "address"     (I-D.maldant-spice-oidc-cwt TBD18) /
    "age_over_21"
  ],
  8: {      / iek /
    1: 2,   / kty : "EC2" /
    -1: 1,  / crv: "P-256" /
    -2: h'62b700b88e612e009145f20c29e27ddbbea5775b09ee80fc97e72d63' +
        h'8cdf744b', / x /
    -3: h'2417935a88f5a360dbc1fc38a176dfd5a41cb94cef3614596e01a887' +
        h'6ba1efe3'  / y /
  },
  9: {      / hpk /
    1: 2,   / kty: "EC2" /
    -1: 1,  / crv: "P-256" /
    -2: h'f5feb97ce9f0427900cfd5e1d21ac3aadfd9b3816cbbe6fdc28321d1' +
        h'5dbb251a', / x /
    -3: h'c780a0f21d6aef7f25ae782e1afe5791a7564f9f6d7c5cad94c3b785' +
        h'524c7bf4'  / y /
  },
  10: -9    / hpa: "ESP256" (I-D.ietf-jose-fully-specified-algorithms TBD-9) /
}
```

| Figure: Issuer Header (SU-ES256, CBOR)

```
[ / payloads    /
  / iat         / 171452160,
  / exp         / 171719999,
  / family_name / "Doe",
  / given_name  / "Jay",
  / email       / "jaydoe@example.org",
  / address     / {
    / formatted / 1: "1234 Main St.\nAnytown, CA 12345\nUSA",
    / street    / 2: "1234 Main St.",
    / locality  / 3: "Anytown",
    / region    / 4: "CA",
    / post code / 5: "90210",
    / country   / 6: "USA"
  },
  / age_over_21 / true
]
```

| Figure: Issuer Payloads (as CBOR array)

When signed and serialized, the CPT is represented by the following CBOR (in hex):

```
8358cfa7010103140576687474470733a2f2f6973737565722e6578616d706c65
0687060418aa18ab18b318bb6b6167655f6f7665725f323108a4010220012158
2062b700b88e612e009145f20c29e27ddbbea5775b09ee80fc97e72d638cdf74
4b2258202417935a88f5a360dbc1fc38a176dfd5a41cb94cef3614596e01a887
6ba1efe309a401022001215820f5feb97ce9f0427900cfd5e1d21ac3aadfd9b3
816cbbe6fdc28321d15dbb251a225820c780a0f21d6aef7f25ae782e1afe5791
a7564f9f6d7c5cad94c3b785524c7bf40a28871a0a3827001a0a3c3d3f63446f
65634a6179726a6179646f65406578616d706c652e6f7267a601782331323334
204d61696e2053742e0a416e79746f776e2c2043412031323334350a55534102
6d31323334204d61696e2053742e0367416e79746f776e046243410565393032
31300663555341f58858406e24dc48da3942b8a46e629c7fc93b5f99fb083f53
397944dbed450a7492015421dfe80ece89ffb522152eb1ec4e6c919438fdb611
9570c19ecc57e38bc5786458409d3fbd702e101be31a47f1607e43fb8baee442
a2253ad765b2b66ad0a95f92b7c01963c18a27c6d6e7512ead5af3cb4a135ef9
dccfccc07a8c78dc63300551715840dbe78e6a2f8a82b8883e94056f385976d2
d6148ac437915b14f63cf8f1f938fa804366f128aba0140ba953652a57562fce
3f5aac98948cf0e23cf02398e287d958405ebbc7dd8ef2d65226cb1d478fcbf4
d17f8b2c2d92736831e5e506cdd6550cef52010fa390caa2d60ca4f7928343bd
987e8e8497ca2455edf785577ba466627d58409b5886130e1c0afe1b769b77fb
eb4adae24a2e01f94224b06abc273621bea28ee729eeb078599a1c0de670e7d5
807c9965a0c4e237341e8a84d276295393afdd58408b4c09b85512d12d906964
3c37f4c6974203424b2463d88ca88fc6fc6f7c636cd4fb01291320146e378f82
106a8ccdb85dd01911df3c7b159a3e84eb593d13395840cf3bb6a429bd3c1f2e
99cac777a07293a974675a7fb6ffd3c19a1cc286a128b0f83c46110c9572d42a
d27790e9e5443bd9cfb988a858d0e5ea42d62182f153305840e89e137a073690
7897b6e7edcfb79e8d98834e2758ab6529794ae031f13c1000f6487ecb178076
b2f8478ccd7be16ef1cf9e796ad73714a9aba917aad27d1cb0
```


| Fixtures: Issued Form (SU-ES256, CBOR)

The presented form, similarly to the issued form above, is made with the holder conveying the same parameters and the same set of selectively disclosed payloads as the JPT above:

```
{          / protected header /
  1: 1,    / alg: "SU-ES256" /
  6: "https://recipient.example.com", / aud /
  7: h’dfc73bb8e4a2f6b6939c09970da15aa16123f7f2502f69ce7e1a87ed8882c0ca’, / nonce /
}
```

| Figure: Presentation Header (SU-ES256, CBOR)

When the appropriate proof is generated, the CPT is serialized into the following CBOR (in hex):

```
845846a3010106781d68747470733a2f2f726563697069656e742e6578616d70
6c652e636f6d075820dfc73bb8e4a2f6b6939c09970da15aa16123f7f2502f69
ce7e1a87ed8882c0ca58cfa70101031405766874747470733a2f2f697373756572
2e6578616d706c650687060418aa18ab18b318bb6b6167655f6f7665725f3231
08a40102200121582062b700b88e612e009145f20c29e27ddbbea5775b09ee80
fc97e72d638cdf744b2258202417935a88f5a360dbc1fc38a176dfd5a41cb94c
ef3614596e01a8876ba1efe309a401022001215820f5feb97ce9f0427900cfd5
e1d21ac3aadfd9b3816cbbe6fdc28321d15dbb251a225820c780a0f21d6aef7f
25ae782e1afe5791a7564f9f6d7c5cad94c3b785524c7bf40a28891a0a382700
1a0a3c3d3f63446f65634a6179726a6179646f65406578616d706c652e6f7267
a6017823313233342044d61696e2053742e0a416e79746f776e2c204341203132
3334350a555341026d31323334204d61696e2053742e0367416e79746f776e04
62434105653930323130066355534 1f5f6f68758406e24dc48da3942b8a46e62
9c7fc93b5f99fb083f53397944dbed450a7492015421dfe80ece89ffb522152e
b1ec4e6c919438fdb6119570c19ecc57e38bc5786458409d3fbd702e101be31a
47f1607e43fb8baee442a2253ad765b2b66ad0a95f92b7c01963c18a27c6d6e7
512ead5af3cb4a135ef9dccfccc07a8c78dc63300551715840dbe78e6a2f8a82
b8883e94056f385976d2d6148ac437915b14f63cf8f1f938fa804366f128aba0
140ba953652a57562fce3f5aac98948cf0e23cf02398e287d958405ebbc7dd8e
f2d65226cb1d478fcbf4d17f8b2c2d92736831e5e506cdd6550cef52010fa390
caa2d60ca4f7928343bd987e8e8497ca2455edf785577ba466627d58409b5886
130e1c0afe1b769b77fbeb4adae24a2e01f94224b06abc273621bea28ee729ee
b078599a1c0de670e7d5807c9965a0c4e237341e8a84d276295393afdd58408b
4c09b85512d12d9069643c37f4c6974203424b2463d88ca88fc6fc6f7c636cd4
fb01291320146e378f82106a8ccdb85dd01911df3c7b159a3e84eb593d133958
40181d1c30ba01716f16a43bf9e7a0bb39deefc4bc427102e06d013f9a274fb0
de5621efdd7c506d6fff78d0368a4d0b9910727bc6deb9ffec68c152bbf75e6b
c2
```


| Figure: Presented Form (SU-ES256, CBOR)

A.3. Example BBS JWP

The following example uses the BBS algorithm.

This is the Issuer’s stable private key in the JWK format:


```
{
  "kty": "OKP",
  "alg": "BBS",
  "use": "proof",
  "crv": "BLS12381G2",
  "x": "jpHvuKcm9_nBaonJxliIQQdrFqk_PpeHEAvU5-2YR7NNH497WkJxMUOKCmgm5
       _H1CF_kuPXlJTs1ij6HSKj6yLb3zIFZyUVANPOL9Wtk37h_d-dS3_-ZCazulTk
       ag7on",
  "d": "I4pwwzNghZ45PlP3pdKrmFZsWYXQGcLxlnEGdH1pVyY"
}
```

Figure 10: BBS private key in JWK format

There is no additional holder key necessary for presentation proofs.

For the following protected header and array of payloads:


```
{
  "kid": "HjfcpyjuZQ-O8Ye2hQnNbT9RbbnrobptdnExR0DUjU8",
  "alg": "BBS"
}
```


Figure 11: Example Issuer Header

These components are signed using the private issuer key previously given, which is then representable in the following serialization:


```
eyJraWQiOiJIamZjcHlqdVpRLU84WWUyaFFuTmJUOVJiYm5yb2JwdGRuRXhSMERValU4I
iwiYWxnIjoiQkJTIn0.MTcxNDUyMTYwMA~MTcxNzE5OTk5OQ~IkRvZSI~IkpheSI~Imph
eWRvZUBleGFtcGxlLm9yZyI~eyJmb3JtYXR0ZWQiOiIxMjM0IE1haW4gU3QuXG5Bbnl0b
3duLCBDQSAxMjM0NVxuVVNBIiwic3RyZWV0X2FkZHJlc3MiOiIxMjM0IE1haW4gU3QuIi
wibG9jYWxpdHkiOiJBbnl0b3duIiwicmVnaW9uIjoiQ0EiLCJwb3N0YWxfY29kZSI6MTI
zNDUsImNvdW50cnkiOiJVU0EifQ~dHJ1ZQ.qyPOly0YiFRvcsowHooAJsSz2bTA5AShYK
enLalfethb5Wp64HphVt4DKzquDYMEEpkkHLULpo0K2t7Z7cS3L0qAFVCLtNff_CaO1uX
1OTU
```


Figure 12: Issued JWP (BBS, JSON, Compact Serialization)

For a presentation with the following Presentation Header:


```
{
    "alg": "BBS",
    "aud": "https://recipient.example.com",
    "nonce": "wrmBRkKtXjQ"
}
```


Figure 13: Presentation Header

The holder decides to share all information other than the email address, and generates a proof. That proof is represented in the following serialization:

```
eyJhbGciOiJCQlMiLCJhdWQiOiJodHRwczovL3JlY2lwaWVudC5leGFtcGxlLmNvbSIsI
m5vbmNlIjoid3JtQlJrS3RYalEifQ.eyJraWQiOiJIamZjcHlqdVpRLU84WWUyaFFuTmJ
UOVJiYm5yb2JwdGRuRXhSMERValU4IiwiYWxnIjoiQkJTIn0.MTcxNDUyMTYwMA~MTcxN
zE5OTk5OQ~IkRvZSI~IkpheSI~~~.iu_WTzqcjO4KaW3Fr2N1ahO9Uo7PHnkOf130XFpW
xEEzyMMl3qKg67rGTHQrLYFdos3E-cjs7IEbOwZYaFNI3ho3kDyDhdIIdukSEoeecDW7c
f6rhhFFe-L-dX4t_eN0rGI5kvxpLBnT6lEXP4y53TdmghkjBU69QnqnoAIT0OleHszORr
DWgIPTQtgDerUXSm4E7r3cR8UuS92C1FQy7F9BX_cmuG8qdm3Cyr7nNQgevI7QpMdMp_T
4tkUKmZzzeTuAdxFV6evO1uhKmT2Dwk6go5orjC14GBB4niIwjG8ItA345t7fDiZ41s1N
IMAgWmtoQRJbVIsAoKxAKBhL8F2AjBDfssfEMbU6JsyWJesQtdQTcmRVPufMKPIaN6e84
UZkjkFHyh8yGONdhB4VnhreybqvJjSH5CQDTD59e4n52vaDMT502SojTNcwvoWcPrQ37k
UE4zifzZ9xxfBmHw4BrWvfrKZdhDs5QFAdjyU
```


Figure 14: Presentation JWP (BBS, JSON, Compact serialization)

A.4. Example MAC JWP

The following example uses the MAC-H256 algorithm.

This is the Issuer’s stable private key in the JWK format:


```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "MQGKGfEcKSXsHBhBSrDjYxvA6NqBh4X5ylt9rGPVuLg",
  "y": "JVguv8zBJUEsNqMAAlOqX_DigKPY4P3oR-LFCqOAvQ0",
  "d": "wilgoSc8vpMoxz8tJ4roRF9di9l9L4RukWrPT6wdZi8"
}
```


Figure 15: Issuer private key

This is the Issuer’s ephemerally generated shared secret:


```
"E-W74PlKrVagSBrrRPztEMlYIGGD9F55PBt-pL1O8D8"
```


Figure 16: Shared Secret

This is the Holder’s presentation private key in the JWK format:


```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "9f65fOnwQnkAz9Xh0hrDqt_Zs4Fsu-b9woMh0V27JRo",
  "y": "x4Cg8h1q738lrnguGv5XkadWT59tfFytlMO3hVJMe_Q",
  "d": "T0hKcZ1iWLGrRSRzC9WZM3xAso9djnpYVNVeu3nql6A"
}
```


Figure 17: Holder private key

For the following Header and array of payloads:


```
{
  "alg": "MAC-H256",
  "hpa": "ES256",
  "typ": "JPT",
  "iss": "https://issuer.example",
  "claims": [
    "iat",
    "exp",
    "family_name",
    "given_name",
    "email",
    "address",
    "age_over_21"
  ],
  "hpk": {
    "kty": "EC",
    "crv": "P-256",
    "use": "sign",
    "x": "9f65fOnwQnkAz9Xh0hrDqt_Zs4Fsu-b9woMh0V27JRo",
    "y": "x4Cg8h1q738lrnguGv5XkadWT59tfFytlMO3hVJMe_Q"
  }
}
```


Figure 18: Example Issuer Header


```
[
    1714521600,
    1717199999,
    "Doe",
    "Jay",
    "jaydoe@example.org",
    {
        "formatted": "1234 Main St.\nAnytown, CA 12345\nUSA",
        "street_address": "1234 Main St.",
        "locality": "Anytown",
        "region": "CA",
        "postal_code": 12345,
        "country": "USA"
    },
    true
]
```


Figure 19: Example issuer payloads (as members of a JSON array)

The issuer generates an array of derived keys, one per payload slot. This is done using the shared secret as the key and a binary value based on the payload slot index (from zero) as input to the HMAC operation.

This results in the following set of derived keys:


```
[
  "jHGnbygUYI5VdO_7DvLq0WRTNSW_uhOax1Y17IUlRAw",
  "0GAdidOeuJ5MgpdMnIagOfa6XfM3Z0NxPtl0x_feV-M",
  "4Jkdr6wXRTc2Q0Ghp8a1Ry4FXIcRqp2Q--3AJW3lCi8",
  "E8zCUoLg7eSsm94sl0D6ND_21mWhQz-BCQ0vmLBruyw",
  "pYGZoTKwJuCt-eihevdercNP9wF6mu4m8DkaIi-9BKc",
  "5tZsuRtnjbZRh2NkwbWNteMfTzm0bm---PhaANFhoi4",
  "2M0w4mF_gUeEQad1h35cervDxeVqniL9r3z0wI8tC04"
]
```


Figure 20: Derived payload keys (Base64url-Encoded)

A MAC is generated for each payload using the corresponding derived payload key. This results in the following set of MAC values:


```
[
  "eDQzGY_TspeKEiFFBYGlzRoPbkg7f62uHvzrBuNdbUQ",
  "yn9iTOJPLqzAPzAMnfWFKLJp76aYg0kPpqAWNzoD5WA",
  "Og-TwlajwiV1zaYttoTq8yAEK_km-TJQkC8FcortoUg",
  "wKFQhIyjXu5Rk9kmV4VrqkdYfAN0oj-cWvzFgZyuvLo",
  "i3Ef-YD065WKUg6pPA3AKR0tJasGOvI_1NTCfHvyclM",
  "qcAhk83ldqom114iktYR9UTSvsecFsCU0q8a-fdnxDU",
  "WxLURctRhf1z5BRtRDD8GHcGlQvJA6RjttDKJox4koo"
]
```


Figure 21: Payload MAC values (Base64url-Encoded)

The Issuer Header and payload MAC values are combined into a binary representation known as the Compact MAC Representation. This representation is signed with the issuer’s private key.

The proof consists of two octet string values: the signature over the combined MAC representation, and the shared secret.


```
[
  "iPvaOPSOUvKrAmDgK1HrgnV889JuSgte_q_Klcq2ltdYAQ53ENjSWbv1eNTopP17D_
Xf2uCW3HWRYELDG4Vl_w",
  "0GvOqgngJ0zsuJWl4NV20mbFxVkzJbfxoRPFqRNLqFQ"
]
```


Figure 22: Issued Proof (Base64url-Encoded)

The final issued JWP in compact serialization is:


```
eyJhbGciOiJNQUMtSDI1NiIsImhwYSI6IkVTMjU2IiwidHlwIjoiSlBUIiwiaXNzIjoia
HR0cHM6Ly9pc3N1ZXIuZXhhbXBsZSIsImNsYWltcyI6WyJpYXQiLCJleHAiLCJmYW1pbH
lfbmFtZSIsImdpdmVuX25hbWUiLCJlbWFpbCIsImFkZHJlc3MiLCJhZ2Vfb3Zlcl8yMSJ
dLCJocGsiOnsia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIsInVzZSI6InNpZ24iLCJ4Ijoi
OWY2NWZPbndRbmtBejlYaDBockRxdF9aczRGc3UtYjl3b01oMFYyN0pSbyIsInkiOiJ4N
ENnOGgxcTczOGxybmd1R3Y1WGthZFdUNTl0ZkZ5dGxNTzNoVkpNZV9RIn19.MTcxNDUyM
TYwMA~MTcxNzE5OTk5OQ~IkRvZSI~IkpheSI~ImpheWRvZUBleGFtcGxlLm9yZyI~eyJm
b3JtYXR0ZWQiOiIxMjM0IE1haW4gU3QuXG5Bbnl0b3duLCBDQSAxMjM0NVxuVVNBIiwic
3RyZWV0X2FkZHJlc3MiOiIxMjM0IE1haW4gU3QuIiwibG9jYWxpdHkiOiJBbnl0b3duIi
wicmVnaW9uIjoiQ0EiLCJwb3N0YWxfY29kZSI6MTIzNDUsImNvdW50cnkiOiJVU0EifQ~
dHJ1ZQ.iPvaOPSOUvKrAmDgK1HrgnV889JuSgte_q_Klcq2ltdYAQ53ENjSWbv1eNTopP
17D_Xf2uCW3HWRYELDG4Vl_w~0GvOqgngJ0zsuJWl4NV20mbFxVkzJbfxoRPFqRNLqFQ
```


Figure 23: Issued JWP (MAC-H256, JSON, Compact Serialization)

Next, we show the presentation of the JWP with selective disclosure.

For presentation with the following Presentation Header:


```
{
  "alg": "MAC-H256",
  "aud": "https://recipient.example.com",
  "nonce": "38c7uOSi9raTnAmXDaFaoWEj9_JQL2nOfhqH7YiCwMo"
}
```


Figure 24: Presentation Header

The holder will take the issuer proof (including shared secret) and derive the same individual payload MAC values (above).

In this case, the holder has decided not to disclose the last three claims provided by the issuer (corresponding to email, address, and age_over_21)

For each payload slot, the holder will provide one of two values as part of the proof value. For a disclosed payload, the holder will provide the corresponding derived key. For a non-disclosed payload, the holder will provide the corresponding MAC value.

The final presented proof value is an array of octet strings. The contents are Presentation Header signature, followed by the issuer signature, then the value disclosed by the holder for each payload. This results in the following proof:

```
[
  "iPvaOPSOUvKrAmDgK1HrgnV889JuSgte_q_Klcq2ltdYAQ53ENjSWbv1eNTopP17D_
Xf2uCW3HWRYELDG4Vl_w",
  "jHGnbygUYI5VdO_7DvLq0WRTNSW_uhOax1Y17IUlRAw",
  "0GAdidOeuJ5MgpdMnIagOfa6XfM3Z0NxPtl0x_feV-M",
  "4Jkdr6wXRTc2Q0Ghp8a1Ry4FXIcRqp2Q--3AJW3lCi8",
  "E8zCUoLg7eSsm94sl0D6ND_21mWhQz-BCQ0vmLBruyw",
  "i3Ef-YD065WKUg6pPA3AKR0tJasGOvI_1NTCfHvyclM",
  "qcAhk83ldqom114iktYR9UTSvsecFsCU0q8a-fdnxDU",
  "WxLURctRhf1z5BRtRDD8GHcGlQvJA6RjttDKJox4koo",
  "ffaMuu14qyIJT-aew2gDLwmjk5RrLR7wU9ekJ-nmAnOrC6gQWJWzpMcUWj1sIVn97l
ycCDAnFWVo7LUc3qqCOw"
]
```


Figure 25: Presentation proof (Base64url-Encoded)

The final presented JWP in compact serialization is:


```
eyJhbGciOiJNQUMtSDI1NiIsImF1ZCI6Imh0dHBzOi8vcmVjaXBpZW50LmV4YW1wbGUuY
29tIiwibm9uY2UiOiIzOGM3dU9TaTlyYVRuQW1YRGFGYW9XRWo5X0pRTDJuT2ZocUg3WW
lDd01vIn0.eyJhbGciOiJNQUMtSDI1NiIsImhwYSI6IkVTMjU2IiwidHlwIjoiSlBUIiw
iaXNzIjoiaHR0cHM6Ly9pc3N1ZXIuZXhhbXBsZSIsImNsYWltcyI6WyJpYXQiLCJleHAi
LCJmYW1pbHlfbmFtZSIsImdpdmVuX25hbWUiLCJlbWFpbCIsImFkZHJlc3MiLCJhZ2Vfb
3Zlcl8yMSJdLCJocGsiOnsia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIsInVzZSI6InNpZ2
4iLCJ4IjoiOWY2NWZPbndRbmtBejlYaDBockRxdF9aczRGc3UtYjl3b01oMFYyN0pSbyI
sInkiOiJ4NENnOGgxcTczOGxybmd1R3Y1WGthZFdUNTl0ZkZ5dGxNTzNoVkpNZV9RIn19
.MTcxNDUyMTYwMA~MTcxNzE5OTk5OQ~IkRvZSI~IkpheSI~~~.iPvaOPSOUvKrAmDgK1H
rgnV889JuSgte_q_Klcq2ltdYAQ53ENjSWbv1eNTopP17D_Xf2uCW3HWRYELDG4Vl_w~j
HGnbygUYI5VdO_7DvLq0WRTNSW_uhOax1Y17IUlRAw~0GAdidOeuJ5MgpdMnIagOfa6Xf
M3Z0NxPtl0x_feV-M~4Jkdr6wXRTc2Q0Ghp8a1Ry4FXIcRqp2Q--3AJW3lCi8~E8zCUoL
g7eSsm94sl0D6ND_21mWhQz-BCQ0vmLBruyw~i3Ef-YD065WKUg6pPA3AKR0tJasGOvI_
1NTCfHvyclM~qcAhk83ldqom114iktYR9UTSvsecFsCU0q8a-fdnxDU~WxLURctRhf1z5
BRtRDD8GHcGlQvJA6RjttDKJox4koo~ffaMuu14qyIJT-aew2gDLwmjk5RrLR7wU9ekJ-
nmAnOrC6gQWJWzpMcUWj1sIVn97lycCDAnFWVo7LUc3qqCOw
```


Figure 26: Presented JWP (MAC-H256, JSON, Compact Serialization)

Appendix B. Acknowledgements

This work was incubated in the DIF Applied Cryptography Working Group (https://identity.foundation/working-groups/crypto.html).

We would like to thank Alberto Solavagione for his valuable contributions to this specification.

The BBS examples were generated using the library at https://github.com/mattrglobal/pairing_crypto (https://github.com/mattrglobal/pairing_crypto) .

Appendix C. Document History

[[To be removed from the final specification]]

-12

* IANA Considerations section changes from IANA Early Review
* Updated BLS keys to match BLS Key Representations draft 8

-11

* Change Issuer Protected Header to Issuer Header
* Change Presentation Protected Header and Holder Presentation Header to Presentation Header

-10

* Clarify MAC issuance and presentation using new "payload slot" nomenclature.
* Define a new binary "Presentation Internal Representation" so that the holder signature protects the entire presentation
* Leverage the new "Holder Presentation Algorithm" to allow the holder algorithm to be independent from the signature algorithm used by the issuer
* Redefine computation of the "Combined MAC Representation" to more closely match the new Presentation Internal Representation.
* Change the MAC algorithm to directly sign the binary Combined MAC Representation rather than convert it to a JWS.
* Do not unnecessarily hash the issuer protected header inside the Combined MAC Representation, so that it can provide some manner of domain separation.
* Clarify how verifiers are to generate the Combined MAC Representation from available information.
* Provider step-by-step instructions for verification of a presentation
* Change Proof Key to Issuer Ephemeral Key and Presentation Key to Holder Presentation Key

-09

* Remove JSON serialization
* Added CBOR (CPT) example to the appendix using SU-ES256

-08

* Made some additional references normative.
* Corrected SU-ES256 issuer protected header including private keys

-07

* Changing primary editor
* Update registry template for algorithms to account for integer CBOR labels
* Restylize initial registry entries for readability
* Defer BBS key definition to [I-D.ietf-cose-bls-key-representations]
* Modify example generation to use proof_key and presentation_key names
* Change proof_jwk to proof_key and presentation_jwk to presentation_key to better represent that the key may be JSON or CBOR-formatted.
* Moved the registry for proof_key and presentation_key to JWP where they are defined. Consolidated usage, purpose, and requirements from algorithm usage under these definitions.
* Combined BBS-PROOF into BBS

-06

* Update reference to new repository home
* Fixed #77: Removed vestigial use of presentation_header.
* Correct pjwk to presentation_jwk

-05

* Update of appendix describing MAC-H256 to now also be generated by the build system from a common set of code and templates.
* Update single use algorithm to use an array of octet values rather than requiring splitting an octet buffer into parts during generation of a presentation and during verification.
* Update BBS algorithm description and examples to clarify the proof is an array with a single octet string.
* Update MAC algorithm to use an array of octet values for the proof, rather than requiring splitting an octet buffer into parts.
* Add new section on the Combined MAC Representation to clarify operations are serving to recreate this octet string value.
* Correct reference to the latest BBS draft.
* SU and MAC families now use raw JWA rather than JWS and synthesized headers
* Change algorithms to not use base64url-encoding internally. Algorithms are meant to operate on octets, while base64url-encoding is used to represent those octets in JSON and compact serializations.

-04

* Refactoring figures and examples to be built from a common set across all three documents
* Move single-use example appendix from JWP to JPA
* Change algorithm from BBS-DRAFT-5 to BBS, and from BBS-PROOF-DRAFT-5 to BBS-PROOF
* Update BBS ciphersuite ID to BBS_BLS12381G1_XMD:SHA-256_SSWU_RO_
* Update to draft 5 BLS key representations

-03

* Improvements resulting from a full proofreading.
* Populated IANA Considerations section.
* Updated to use BBS draft -05.
* Updated examples.

-02

* Add new BBS-DRAFT-3 and BBS-PROOF-DRAFT-3 algorithms based on draft-irtf-cfrg-bbs-signatures-03.
* Remove prior BBS-X algorithm based on a particular implementation of earlier drafts.

-01

* Correct cross-references within group
* Describe issuer_header and presentation_header
* Update BBS references to CFRG drafts
* Rework reference to HMAC (RFC2104)
* Remove ZKSnark placeholder

-00

* Created initial working group draft based on draft-jmiller-jose-json-proof-algorithms-01

Authors’ Addresses

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

David Waite
Ping Identity
Email: dwaite+jwp@pingidentity.com

Jeremie Miller
Ping Identity
Email: jmiller@pingidentity.com