Workgroup: jose
Internet-Draft:
draft-ietf-jose-json-web-proof-03
Published: 1 March 2024
Intended Status: Standards Track
Expires: 2 September 2024
Authors: J. Miller
Ping Identity
D. Waite
Ping Identity
M. Jones
Self-Issued Consulting

# JSON Web Proof

## Abstract

The JOSE set of standards established JSON-based container formats for Keys, Signatures, and Encryption. They also established IANA registries to enable the algorithms and representations used for them to be extended. Since those were created, newer cryptographic algorithms that support selective disclosure and unlinkability have matured and started seeing early market adoption.

This document defines a new container format similar in purpose and design to JSON Web Signature (JWS) called a *JSON Web Proof (JWP)*. Unlike JWS, which integrity-protects only a single payload, JWP can integrity-protect multiple payloads in one message. It also specifies a new presentation form that supports selective disclosure of individual payloads, enables additional proof computation, and adds a protected header to prevent replay.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 2 September 2024.

**Table of Contents**

## 1. Introduction

The JOSE specifications are very widely deployed and well supported, enabling use of cryptographic primitives with a JSON representation. JWTs [RFC7519] are one of the most common representations for identity and access claims. For instance, they are used by the OpenID Connect and Secure Telephony Identity Revisited (STIR) standards. Also, JWTs are used by W3C's Verifiable Credentials and are used in many decentralized identity systems.

With these new use cases, there is an increased focus on adopting privacy-protecting cryptographic primitives. While such primitives are still an active area of academic and applied research, the leading candidates introduce new patterns that are not currently supported by JOSE. These new patterns are largely focused on two areas: supporting selective disclosure when presenting information and minimizing correlation through the use of Zero-Knowledge Proofs (ZKPs) in addition to traditional signatures.

There are a growing number of these cryptographic primitives that support selective disclosure while protecting privacy across multiple presentations. Examples used in the context of Verifiable Credentials are:

* CL Signatures
* IDEMIX
* BBS signatures, described in [I-D.irtf-cfrg-bbs-signatures]
* MerkleDisclosureProof2021
* Mercurial Signatures
* PS Signatures
* U-Prove
* Spartan

All of these follow the same pattern of taking multiple claims (a.k.a., "attributes" or "messages" in the literature) and binding them together into a single issued token. These are then later

securely one-way transformed into a presentation that reveals potentially only a subset of the original claims, predicate proofs about the claim values, or proofs of knowledge of the claims.

## 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.

### 2.1. Terminology

The following terms are used throughout this series of documents:

binding: A mechanism, indicated in an issued JWP, for how to verify a presentation was created by the intended holder.

linkability: The property where multiple presentations may be correlated to a single issued JWP, either through consistency in the cryptographic integrity or due to particulars of JWP usage by an application. Such issued JWPs may be referred to as single-use, as multiple uses may leak unintended knowledge.

unlinkability: The property of issuance and presentation algorithms and of application usage, where one presentation can only be correlated with other presentations based on holder-disclosed information.

### 2.2. Abbreviations

*ZKP: Zero-Knowledge Proof
*JWP: JSON Web Proof (this specification)
*JPA: JSON Proof Algorithms [I-D.ietf-jose-json-proof-algorithms]
*JPT: JSON Proof Token [I-D.ietf-jose-json-proof-token]

## 3. Background

A *JSON Web Proof (JWP)* is very similar to a JWS [RFC7515], with the addition that it can contain multiple individual secured payloads instead of a single one. JWP-supporting algorithms are then able to separate and act on the individual payloads contained within.

The intent of JSON Web Proofs is to establish a common container format for multiple payloads that can be integrity-verified against a cryptographic proof value also in the container. It does not create or specify any cryptographic protocols, multi-party protocols, or detail any algorithm-specific capabilities.

To fully support the newer privacy primitives, JWP utilizes the three roles of issuer, holder, and verifier, as defined by the VC Data Model [VC-DATA-MODEL-2.0]. There are also two forms of a JWP: the issued form created by an issuer for a holder, and the presented form created by a holder for a verifier.

The four principal interactions used by JWP are issue, confirm, present, and verify.

A JWP is initially created by the issuer using the issue interaction. A successful result is an issued JWP that has a single issuer-protected header, one or more payloads, and an initial proof value that contains the issuing algorithm output. The holder, upon receiving an issued JWP, then uses the confirm interaction to check the integrity protection of the header and all payloads using the proof value.

After validation, the holder uses the present interaction to apply any selective disclosure choices, perform privacy-preserving transformations for unlinkability, and add a presentation-protected header that ensures the resulting presented JWP cannot be replayed. The verifier then uses the verify interaction to ensure the integrity protection of the protected headers and any disclosed payloads, along with verifying any additional ZKPs covering non-disclosed payloads.

While issue and confirm only occur when a JWP is initially created by the issuer, the present and verify steps may be safely repeated by a holder on an issued JWP. He resulting presented JWP is only unlinkable when supported by the underlying algorithm.

Algorithm definitions that support JWPs are in separate companion specifications - just as the JSON Web Algorithms [RFC7518] specification does for JWS and JWE [RFC7516]. The JSON Proof Algorithms (JPA) [I-D.ietf-jose-json-proof-algorithms] specification defines how an initial set of algorithms are used with JWP.

## 4. JWP Header

The members of the JSON object(s) representing the JWP Header describe the proof applied to the Protected Header and the Payload and optionally, additional properties of the JWP. The Header Parameter names within the JWP Header MUST be unique; JWP parsers MUST either reject JWPs with duplicate Header Parameter names or use

a JSON parser that returns only the lexically last duplicate member name, as specified in Section 15.12 ("The JSON Object") of ECMAScript 5.1 [ECMAScript].

Implementations are required to understand the specific Header Parameters defined by this specification that are designated as "MUST be understood" and process them in the manner defined in this specification. All other Header Parameters defined by this specification that are not so designated MUST be ignored when not understood. Unless listed as a critical Header Parameter, per Section 4.1.4, all Header Parameters not defined by this specification MUST be ignored when not understood.

There are three classes of Header Parameter names: Registered Header Parameter names, Public Header Parameter names, and Private Header Parameter names.

These requirements are intentionally parallel to those in Section 4 of [RFC7515].

### 4.1. Registered Header Parameter Names

The following Header Parameter names for use in JWPs are registered in the IANA "JSON Web Proof Header Parameters" registry established by Section 8.1, with meanings as defined in the subsections below.

As indicated by the common registry, Header Parameters used in the Issued Form (see Section 5.1) and the Presented Form Section 5.2 share a common Header Parameter space; when a parameter is used by both forms, its usage must be compatible between them.

These Header Parameters are intentionally parallel to those in Section 4.1 of [RFC7515].

#### 4.1.1. "alg" (Algorithm) Header Parameter

The alg (algorithm) Header Parameter identifies the cryptographic algorithm used to secure the JWP. The JWP Proof value is not valid if the alg value does not represent a supported algorithm or if there is not a key for use with that algorithm associated with the party that secured the content. alg values should either be registered in the IANA "JSON Web Proof Algorithms" registry established by [I-D.ietf-jose-json-proof-algorithms] or be a value that contains a Collision-Resistant Name. The alg value is a case-sensitive ASCII string containing a StringOrURI value. This Header Parameter MUST be present and MUST be understood and processed by implementations.

A list of defined alg values for this use can be found in the IANA "JSON Web Proof Algorithms" registry established by

[I-D.ietf-jose-json-proof-algorithms]; the initial contents of this registry are registered by [I-D.ietf-jose-json-proof-algorithms].

#### 4.1.2. "kid" (Key ID) Header Parameter

The kid (key ID) Header Parameter is a hint indicating which key was used to secure the JWP. This parameter allows originators to explicitly signal a change of key to recipients. The structure of the kid value is unspecified. Its value MUST be a case-sensitive string. Use of this Header Parameter is OPTIONAL.

When used with a JWK, the kid value is used to match a JWK kid parameter value.

#### 4.1.3. "typ" (Type) Header Parameter

The typ (type) Header Parameter is used by JWP applications to declare the media type (#IANA.MediaTypes) of this complete JWP. This is intended for use by the application when more than one kind of object could be present in an application data structure that can contain a JWP; the application can use this value to disambiguate among the different kinds of objects that might be present. It will typically not be used by applications when the kind of object is already known. This parameter is ignored by JWP implementations; any processing of this parameter is performed by the JWP application. Use of this Header Parameter is OPTIONAL.

Per [RFC2045], all media type values, subtype values, and parameter names are case insensitive. However, parameter values are case sensitive unless otherwise specified for the specific parameter.

To keep messages compact in common situations, it is RECOMMENDED that producers omit an "application/" prefix of a media type value in a typ Header Parameter when no other '/' appears in the media type value. A recipient using the media type value MUST treat it as if "application/" were prepended to any typ value not containing a '/'. For instance, a typ value of example SHOULD be used to represent the application/example media type, whereas the media type application/example;part="1/2" cannot be shortened to example;part="1/2".

The typ value jwp can be used by applications to indicate that this object is a JWP using the JWP Compact Serialization. The typ value jwp+json can be used by applications to indicate that this object is a JWP using the JWP JSON Serialization. Other type values can also be used by applications.

It is RECOMMENDED that the typ Header Parameter be used for explicit typing, in parallel to the recommendations in Section 3.11 of [RFC8725].

#### 4.1.4. "crit" (Critical) Header Parameter

The crit (critical) Header Parameter indicates that extensions to this specification and/or [I-D.ietf-jose-json-proof-algorithms] are being used that MUST be understood and processed. Its value is an array listing the Header Parameter names present in the JWP Header that use those extensions. If any of the listed extension Header Parameters are not understood and supported by the recipient, then the JWP is invalid. Producers MUST NOT include Header Parameter names defined by this specification or [I-D.ietf-jose-json-proof-algorithms] for use with JWP, duplicate names, or names that do not occur as Header Parameter names within the JWP Header in the crit list. Producers MUST NOT use the empty list [] as the crit value. Recipients MAY consider the JWP to be invalid if the critical list contains any Header Parameter names defined by this specification or [I-D.ietf-jose-json-proof-algorithms] for use with JWP or if any other constraints on its use are violated. When used, this Header Parameter MUST be integrity protected; therefore, it MUST occur only within the JWP Protected Header. Use of this Header Parameter is OPTIONAL. This Header Parameter MUST be understood and processed by implementations.

#### 4.1.5. "proof_jwk" (Proof JWK) Header Parameter

The proof_jwk (Proof JWK) represents the public key used by the issuer for proof of possession. This key is represented as a JSON Web Key public key value. It MUST contain only public key parameters and SHOULD contain only the minimum JWK parameters necessary to represent the key; other JWK parameters included can be checked for consistency and honored, or they can be ignored. This Header Parameter MUST be present in the JWP issuer header parameters and MUST be understood and processed by implementations.

#### 4.1.6. "presentation_jwk" (Presentation JWK) Header Parameter

The presentation_jwk (Presentation JWK) represents the public key used by the holder for proof of possession. This key is represented as a JSON Web Key public key value. It MUST contain only public key parameters and SHOULD contain only the minimum JWK parameters necessary to represent the key; other JWK parameters included can be checked for consistency and honored, or they can be ignored. This Header Parameter MUST be present in the JWP issuer header parameters and MUST be understood and processed by implementations.

#### 4.1.7. "iss" (Issuer) Header Parameter

The iss (issuer) Header Parameter identifies the principal that issued the JWP. The processing of this claim is generally

application specific. The iss value is a case-sensitive string containing a StringOrURI value. Its definition is intentionally parallel to the iss claim defined in [RFC7519]. Use of this Header Parameter is OPTIONAL.

#### 4.1.8. "aud" (Audience) Header Parameter

The aud (audience) Header Parameter identifies the recipients that the JWP is intended for. Each principal intended to process the JWP MUST identify itself with a value in the audience Header Parameter. If the principal processing the Header Parameter does not identify itself with a value in the aud Header Parameter when this Header Parameter is present, then the JWP MUST be rejected. In the general case, the aud value is an array of case-sensitive strings, each containing a StringOrURI value. In the special case when the JWP has one audience, the aud value MAY be a single case-sensitive string containing a StringOrURI value. The interpretation of audience values is generally application specific. Its definition is intentionally parallel to the aud claim defined in [RFC7519]. Use of this Header Parameter is OPTIONAL.

#### 4.1.9. "nonce" (Nonce) Header Parameter

The nonce (nonce) Header Parameter is a case-sensitive string value used to associate protocol state with a JWP. This can be used, for instance, to mitigate replay attacks. The use of nonce values is generally protocol specific. Its definition is intentionally parallel to the nonce claim registered in the IANA "JSON Web Token Claims" registry (#IANA.JWT.Claims). Use of this Header Parameter is OPTIONAL.

#### 4.1.10. "claims" (Claims) Header Parameter

The claims Header Parameter is an array listing the Claim Names corresponding to the JWP payloads, in the same order as the payloads. Each array value is a Claim Name, as defined in [RFC7519]. Use of this Header Parameter is OPTIONAL.

### 4.2. Public Header Parameter Names

Additional Header Parameter names can be defined by those using JWPs. However, in order to prevent collisions, any new Header Parameter name should either be registered in the IANA "JSON Web Proof Header Parameters" registry established by Section 8.1 or be a Public Name (a value that contains a Collision-Resistant Name). In each case, the definer of the name or value needs to take reasonable precautions to make sure they are in control of the part of the namespace they use to define the Header Parameter name.

New Header Parameters should be introduced sparingly, as they can result in non-interoperable JWPs.

### 4.3. Private Header Parameter Names

A producer and consumer of a JWP may agree to use Header Parameter names that are Private Names (names that are not Registered Header Parameter names Section 4.1 or Public Header Parameter names Section 4.2. Unlike Public Header Parameter names, Private Header Parameter names are subject to collision and should be used with caution.

## 5. JWP Forms

A JWP is always in one of two forms: the issued form or the presented form. A structural difference between the two forms is the number of protected headers. An issued JWP has only one issuer protected header, while a presented JWP will have both the issuer protected header and an additional presentation protected header. Each protected header is a JSON object that is serialized as a UTF-8 encoded octet string.

All JWP forms have one or more payloads; each payload is an octet string. The payloads are arranged in an array for which the ordering is preserved in all serializations.

The JWP proof value is one or more octet strings that are only meant to be generated from and processed by the underlying JPA. Internally, the proof value may contain one or more cryptographic statements that are used to check the integrity protection of the header(s) and all payloads. Each of these statements may be a ZKP or a traditional cryptographic signature. The algorithm is responsible for how these statements are serialized into a single proof value.

### 5.1. Issued Form

When a JWP is first created, it is always in the issued form. It will contain the issuer protected header along with all of the payloads.

The issued form can only be confirmed by a holder as being correctly formed and protected. It is NOT to be verified directly or presented as-is to a verifier. The holder SHOULD treat an issued JWP as private and use appropriately protected storage.

#### 5.1.1. Issuer Protected Header

The issuer protected header applies to all of the payloads equally. It is recommended that any payload-specific information not be included in this header and instead be handled outside of the

cryptographic envelope. This is to minimize any correlatable signals in the metadata, to reduce a verifier's ability to group different presentations based on small header variations from the same issuer. The protected header is always disclosed, whereas payloads can be selectively disclosed.

Every issuer protected header MUST have an alg value that identifies a valid JSON Proof Algorithm (JPA).

For example:

```
{
    "alg":"BBS-DRAFT-5"
}
```

#### 5.1.2. Issuer Payloads

Payloads are represented and processed as individual octet strings and arranged in an ordered array when there are multiple payloads. All application context of the placement and encoding of each payload value is out of scope of this specification and SHOULD be well defined and documented by the application or other specifications.

JPAs MAY provide software interfaces that perform the encoding of individual payloads which accept native inputs such as numbers, sets, or elliptic curve points. This enables the algorithm to support advanced features such as blinded values and predicate proofs. These interfaces would generate the octet string encoded payload value as well as include protection of that payload in the combined proof value.

#### 5.1.3. Issuer Proof

The proof value is one or more binary octet strings that are opaque to applications. Individual proof-supporting algorithms are responsible for the contents and security of the proof value, along with any required internal structures.

The issuer proof is used by the holder to perform validation, checking that the issuer header and all payloads are properly encoded and protected by the given proof.

### 5.2. Presented Form

When an issued JWP is presented, it undergoes a transformation that adds a presentation protected header. It may also have one or more payloads hidden, disclosing only a subset of the original issued payloads. The proof value will always be updated to add integrity

protection of the presentation header along with the necessary cryptographic statements to verify the presented JWP.

When supported by the underling JPA, a single issued JWP can be used to safely generate multiple presented JWPs without becoming correlatable.

A JWP may also be single use, where an issued JWP can only be used once to generate a presented form. In this case, any additional presentations would be inherently correlatable. These are still useful for applications needing only selective disclosure or where new unique issued JWPs can be retrieved easily.

#### 5.2.1. Presentation Protected Header

The presented form of a JWP MUST contain a presentation protected header. It is added by the holder and MUST be integrity protected by the underling JPA.

This header is used to ensure that a presented JWP cannot be replayed and is cryptographically bound to the verifier it was presented to.

While there are not any required values in the presentation header, it MUST contain one or more header values that uniquely identify the presented JWP to both the holder and verifier. For example, header values that would satisfy this requirement include nonce and aud.

#### 5.2.2. Presentation Payloads

Any one or more payloads may be non-disclosed in a presented JWP. When a payload is not disclosed, the position of other payloads does not change; the resulting array will simply be sparse and only contain the disclosed payloads.

The disclosed payloads will always be in the same array positions to preserve any index-based references by the application between the issued and presented forms of the JWP. How the sparse array is represented is specific to the serialization used.

Algorithms MAY support including a proof about a payload in the presentation. Applications then treat that proven payload the same as any other non-disclosed payload and do not include it in the presented array of payloads. Rather, proofs about payloads, such as "age >= 21", are included in the presentation proof.

### 5.2.3. Presentation Proof

The proof value of a presented JWP will always be different than the issued proof. At a minimum, it MUST be updated to include protection of the added presentation header.

Algorithms SHOULD generate an un-correlatable presentation proof in order to support multiple presentations from a single issued JWP.

Any payload specific proofs are included in the single proof value for the presented JWP. The JPA is responsible for internally encoding multiple proof values into one and cryptographically binding them to a specific payload from the issuer.

## 6. Serializations

Each disclosed payload MUST be base64url encoded when preparing it to be serialized. The headers and proof are also individually base64url encoded.

Like JWS, JWP supports both a Compact Serialization and a JSON Serialization.

### 6.1. Compact Serialization

The individually encoded payloads are concatenated with the ~ character to form an ordered delimited array. Any non-disclosed payloads are left blank, resulting in sequential ~~ characters such that all payload positions are preserved.

A payload which is disclosed but which contains no data (i.e. a zero-length octet string) is encoded as a single _ character of data, which is not a valid result from base64url-encoding a value.

Additionally, an algorithm MAY supply multiple octet strings for a proof. These are concatenated with the ~ character to form an ordered delimited array.

The headers, concatenated payloads, and proof value are then concatenated with a . character to form the final compact serialization. The issued form will only contain one header and always have three . separated parts. The presented form will always have four . separated parts, the issued header, followed by the protected header, then the payloads and the proof.

### 6.2. JSON Serialization

Non-disclosed payloads in the JSON serialization are represented with a null value in the payloads array. A zero-length payload is

represented as a zero-length base64url encoded sequence, the empty string "".

Proofs are represented as an array of one or more encoded octet strings.

This example flattened JSON serialization shows the presentation form with both the issuer and presentation headers, and with the first and third payloads hidden.


```
{
  "payloads": [
    null,
    "IkpheSI",
    null,
    "NDI"
  ],
  "issuer": "eyJpc3MiOiJodHRwczovL2lzc3Vlci50bGQiLCJjbGFpbXMiOlsiZmFt
  aWx5X25hbWUiLCJnaXZlbl9uYW1lIiwiZW1haWwiLCJhZ2UiXSwidHlwIjoiSlBUIiw
  icHJvb2ZfandrIjp7ImNydiI6IlAtMjU2Iiwia3R5IjoiRUMiLCJ4IjoiYWNiSVFpdU
  1zM2k4X3VzekVqSjJ0cFR0Uk00RVUzeXo5MVBINkNkSDJWMCIsInkiOiJfS2N5TGo5d
  ldNcHRubUt0bTQ2R3FEejh3Zjc0STVMS2dybDJHekgzblNFIn0sInByZXNlbnRhdGlv
  bl9qd2siOnsiY3J2IjoiUC0yNTYiLCJrdHkiOiJFQyIsIngiOiJvQjFUUHJFX1FKSUw
  2MWZVT09LNURwS2dkOGoyemJaSnRxcElMRFRKWDZJIiwieSI6IjNKcW5ya3VjTG9ia2
  RSdU9xWlhPUDlNTWxiRnllbkZPTHlHbEctRlBBQ00ifSwiYWxnIjoiU1UtRVMyNTYif
  Q",
  "proof": [
    "LJMiN6caEqShMJ5jPNts8OescqNq5vKSqkfAdSuGJA1GyJyyrfjkpAG0cDJKZoUg
    omHu5MzYhTUsa0YRXVBnMB91RjonrnWVsakfXtfm2h7gHxA_8G1wkB09x09kon2eK
    9gTv4iKw4GP6Rh02PEIAVAvnhtuiShMnPqVw1tCBdhweWzjyxJbG86J7Y8MDt2H9f
    5hhHIwmSLwXYzCbD37WmvUEQ2_6whgAYB5ugSQN3BjXEviCA__VX3lbhH1RVc27EY
    kRHdRgGQwWNtuExKz7OmwH8oWizplEtjWJ5WIlJpee79gQ9HTa2QIOT9bUDvjjkkO
    -jK_zuDjZwh5MkrcaQ"
  ],
  "presentation": "eyJub25jZSI6InVURUIzNzFsMXB6V0psN2FmQjB3aTBIV1VOaz
  FMZS1iQ29tRkx4YThLLXMifQ"
  ]
}
```


Figure 1: jwp-final-presentation

## 7. Security Considerations

Notes to be expanded:

- *Requirements for supporting algorithms, see JPA
- *Application interface for verification
- *Data minimization of the protected header

*To prevent accidentally introducing linkability, when an issuer uses the same key with the same grouping of payload types, they SHOULD also use the same issuer protected header. Each of these headers SHOULD have the same base64url-serialized value to avoid any non-deterministic JSON serialization.

## 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 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 header parameter: 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 mailing list) for resolution.

Criteria that should be applied by the Designated Experts includes 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 Header Parameters Registry

This specification establishes the IANA "JSON Web Proof Header Parameters" registry for Header Parameter names. The registry records the Header Parameter name and a reference to the specification that defines it. The same Header Parameter name can be registered multiple times, provided that the parameter usage is compatible between the specifications. Different registrations of the same Header Parameter name will typically use different Header Parameter Usage Locations values.

#### 8.1.1. Registration Template

- Header Parameter Name: The name requested (e.g., "kid"). Because a core goal of this specification is for the resulting representations to be compact, it is RECOMMENDED that the name be short -- not to exceed 8 characters without a compelling reason to do so. This name is case sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.
- Header Parameter Description: Brief description of the Header Parameter (e.g., "Key ID").
- Header Parameter Usage Location(s): The Header Parameter 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.
- Change Controller: For Standards Track 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.

#### 8.1.2. Initial Registry Contents

This section registers the Header Parameter names defined in Section 4.1 in this registry.

- Header Parameter Name: alg
- Header Parameter Description: Algorithm
- Header Parameter Usage Location(s): Issued, Presented
- Change Controller: IETF
- Specification Document(s): Section 4.1.1 of this specification

*Header Parameter Name: kid

*Header Parameter Description: Key ID

*Header Parameter Usage Location(s): Issued, Presented

*Change Controller: IETF

*Specification Document(s): Section 4.1.2 of this specification

*Header Parameter Name: typ

*Header Parameter Description: Type

*Header Parameter Usage Location(s): Issued, Presented

*Change Controller: IETF

*Specification Document(s): Section 4.1.3 of this specification

*Header Parameter Name: crit

*Header Parameter Description: Critical

*Header Parameter Usage Location(s): Issued, Presented

*Change Controller: IETF

*Specification Document(s): Section 4.1.4 of this specification

*Header Parameter Name: iss

*Header Parameter Description: Issuer

*Header Parameter Usage Location(s): Issued, Presented

*Change Controller: IETF

*Specification Document(s): Section 4.1.7 of this specification

*Header Parameter Name: aud

*Header Parameter Description: Audience

*Header Parameter Usage Location(s): Presented

*Change Controller: IETF

*Specification Document(s): Section 4.1.8 of this specification

*Header Parameter Name: nonce

*Header Parameter Description: Nonce

*Header Parameter Usage Location(s): Presented

*Change Controller: IETF

*Specification Document(s): Section 4.1.9 of this specification

*Header Parameter Name: claims

*Header Parameter Description: claims

*Header Parameter Usage Location(s): Issued

*Change Controller: IETF

*Specification Document(s): Section 4.1.10 of this specification

## 8.2. Media Type Registration

### 8.2.1. Registry Contents

This section registers the application/jwp media type [RFC2046] in the IANA "Media Types" registry (#IANA.MediaTypes) in the manner described in [RFC6838], which can be used to indicate that the content is a JWP using the JWP Compact Serialization. This section also registers the application/jwp+json media type in the IANA "Media Types" registry, which can be used to indicate that the content is a JWP using the JWP JSON Serialization.

*Type name: application

*Subtype name: jwp

*Required parameters: n/a

*Optional parameters: n/a

*Encoding considerations: 8bit; application/jwp values are encoded as a series of base64url-encoded values (some of which may be the empty string), each separated from the next by a single period ('.') character.

*Security considerations: See the Security Considerations section of this specification.

*Interoperability considerations: n/a

*Published specification: this specification

*Applications that use this media type: TBD

*Fragment identifier considerations: n/a

*Additional information:<list style="empty">

   -Magic number(s): n/a
   -File extension(s): n/a
   -Macintosh file type code(s): n/a

*Person & email address to contact for further information:
 Michael B. Jones, michael_b_jones@hotmail.com

*Intended usage: COMMON

*Restrictions on usage: none

*Author: Michael B. Jones, michael_b_jones@hotmail.com

*Change Controller: IETF

*Provisional registration? No

*Type name: application

*Subtype name: jwp+json

*Required parameters: n/a

*Optional parameters: n/a

*Encoding considerations: 8bit; application/jwp+json values are
 represented as a JSON Object; UTF-8 encoding SHOULD be employed
 for the JSON object.

*Security considerations: See the Security Considerations section
 of this specification

*Interoperability considerations: n/a

*Published specification: this specification

*Applications that use this media type: TBD

*Fragment identifier considerations: n/a

*Additional information:<list style="empty">

   -Magic number(s): n/a
   -File extension(s): n/a
   -Macintosh file type code(s): n/a

*Person & email address to contact for further information:
Michael B. Jones, michael_b_jones@hotmail.com

*Intended usage: COMMON

*Restrictions on usage: none

*Author: Michael B. Jones, michael_b_jones@hotmail.com

*Change Controller: IETF

*Provisional registration? No

# 9. References

## 9.1. Normative References


**[I-D.ietf-jose-json-proof-algorithms]** Miller, J., Jones, M. B., and D. Waite, "JSON Proof Algorithms", Work in Progress, Internet-Draft, draft-ietf-jose-json-proof-algorithms-02, 21 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-jose-json-proof-algorithms-02>.

**[I-D.ietf-jose-json-proof-token]** Miller, J., Jones, M. B., and D. Waite, "JSON Proof Token", Work in Progress, Internet-Draft, draft-ietf-jose-json-proof-token-02, 21 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-jose-json-proof-token-02>.

**[RFC7519]** Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.


## 9.2. Informative References


**[ECMAScript]** Ecma International, "ECMAScript Language Specification, 5.1 Edition", ECMA 262, June 2011, <http://www.ecma-international.org/ecma-262/5.1/ECMA-262.pdf>.

**[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-05, 21 December 2023, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bbs-signatures-05>.

**[RFC2045]** Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

**[RFC2046]** Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

**[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>.

**[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>.

**[RFC6838]** Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

**[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>.

**[RFC7516]** Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

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

**[RFC8174]** Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

**[RFC8725]** Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/info/rfc8725>.

**[VC-DATA-MODEL-2.0]** Sporny, M., Jr, T. T., Herman, I., Jones, M. B., and G. Cohen, "Verifiable Credentials Data Model 2.0", 27 December 2023, <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 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 Signer's stable key and the other is an ephemeral key generated by the Signer just for this JWP.

This is the Signer's stable private key used in this example in the JWK format:


```
{
  "crv": "P-256",
  "kty": "EC",
  "x": "ONebN43-G5DOwZX6jCVpEYEe0bYd5WDybXAG0sL3iDA",
  "y": "b0MHuYfSxu3Pj4DAyDXabAc0mPjpB1worEpr3yyrft4",
  "d": "jnE0-9YvxQtLJEKcyUHU6HQ3Y9nSDnh0NstYJFn7RuI"
}
```


Figure 2: issuer-private-jwk

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


```
{
  "crv": "P-256",
  "kty": "EC",
  "x": "acbIQiuMs3i8_uszEjJ2tpTtRM4EU3yz91PH6CdH2V0",
  "y": "_KcyLj9vWMptnmKtm46GqDz8wf74I5LKgrl2GzH3nSE"
}
```


Figure 3: issuer-ephemeral-jwk

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

```
{
  "crv": "P-256",
  "kty": "EC",
  "x": "oB1TPrE_QJIL61fUOOK5DpKgd8j2zbZJtqpILDTJX6I",
  "y": "3JqnrkucLobkdRuOqZXOP9MMlbFyenFOLyGlG-FPACM"
}
```

Figure 4: holder-presentation-jwk

The JWP Protected 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.

```
{
  "iss": "https://issuer.tld",
  "claims": [
    "family_name",
    "given_name",
    "email",
    "age"
  ],
  "typ": "jpt",
  "proof_jwk": {
    "crv": "P-256",
    "kty": "EC",
    "x": "acbIQiuMs3i8_uszEjJ2tpTtRM4EU3yz91PH6CdH2V0",
    "y": "_KcyLj9vWMptnmKtm46GqDz8wf74I5LKgrl2GzH3nSE"
  },
  "presentation_jwk": {
    "crv": "P-256",
    "kty": "EC",
    "x": "oB1TPrE_QJIL61fUOOK5DpKgd8j2zbZJtqpILDTJX6I",
    "y": "3JqnrkucLobkdRuOqZXOP9MMlbFyenFOLyGlG-FPACM"
  },
  "alg": "SU-ES256"
}
```

Figure 5: jwp-issuer-header

After removing formatting whitespace, the octets representing UTF8(JWP Protected Header) in this example (using JSON array notation) are:

```
[123, 34, 105, 115, 115, 34, 58, 34, 104, 116, 116, 112, 115, 58, 47,
 47, 105, 115, 115, 117, 101, 114, 46, 116, 108, 100, 34, 44, 34, 99,
 108, 97, 105, 109, 115, 34, 58, 91, 34, 102, 97, 109, 105, 108, 121,
 95, 110, 97, 109, 101, 34, 44, 34, 103, 105, 118, 101, 110, 95, 110,
 97, 109, 101, 34, 44, 34, 101, 109, 97, 105, 108, 34, 44, 34, 97,
 103, 101, 34, 93, 44, 34, 116, 121, 112, 34, 58, 34, 74, 80, 84, 34,
 44, 34, 112, 114, 111, 111, 102, 95, 106, 119, 107, 34, 58, 123, 34,
 99, 114, 118, 34, 58, 34, 80, 45, 50, 53, 54, 34, 44, 34, 107, 116,
 121, 35, 58, 34, 69, 67, 34, 44, 34, 120, 34, 58, 34, 97, 99, 98,
 73, 81, 105, 117, 77, 115, 51, 105, 56, 95, 117, 115, 122, 69, 106,
 74, 50,116, 112, 84, 116, 82, 77, 52, 69, 85, 51, 121, 122, 57, 49,
 80, 72, 54, 67, 100, 72, 50, 86, 48, 34, 44, 34, 121, 34, 58, 34,
 95, 75, 99, 121, 76, 106, 57, 118, 87, 77, 112, 116, 110, 109, 75,
 116, 109, 52, 54, 71, 113, 68, 122, 56, 119, 102, 55, 52, 73, 53,
 76, 75, 103, 114, 108, 50, 71, 122, 72, 51, 110, 83, 69, 34, 125,
 44, 34, 112, 114, 101, 115, 101, 110, 116, 97, 116, 105, 111, 110,
 95, 106, 119, 107, 34, 58, 123, 34, 99, 114, 118, 34, 58, 34, 80,
 45, 50, 53, 54, 34, 44, 34, 107, 116, 121, 34, 58, 34, 69, 67, 34,
 44, 34, 120, 34, 58, 34, 111, 66, 49,  84, 80, 114, 69, 95, 81, 74,
 73, 76, 54, 49, 102, 85, 79, 79, 75, 53, 68, 112, 75, 103, 100, 56,
 106, 50, 122, 98, 90, 74, 116, 113, 112, 73, 76, 68, 84, 74, 88, 54,
 73, 34, 44, 34, 121, 34, 58, 34, 51, 74, 113, 110, 114, 107, 117,
 99, 76, 111, 98, 107, 100, 82, 117, 79, 113, 90, 88, 79, 80, 57, 77,
 77, 108, 98, 70, 121, 101, 110, 70, 79, 76, 121, 71, 108, 71, 45,
 70, 80, 65, 67, 77, 34, 125, 44, 34, 97, 108, 103, 34, 58, 34, 83,
 85, 45, 69, 83, 50, 53, 54, 34, 125]
```

Figure 6: jwp-issuer-header-octets

Encoding this JWP Protected Header as BASE64URL(UTF8(JWP Protected Header)) gives this value:

```
eyJpc3MiOiJodHRwczovL2lzc3Vlci50bGQiLCJjbGFpbXMiOlsiZmFtaWx5X25hbWUiL
CJnaXZlbl9uYW1lIiwiZW1haWwiLCJhZ2UiXSwidHlwIjoiSlBUIiwicHJvb2ZfandrIj
p7ImNydiI6IlAtMjU2Iiwia3R5IjoiRUMiLCJ4IjoiYWNiSVFpdU1zM2k4X3VzekVqSjJ
0cFR0Uk00RVUzeXo5MVBINkNkSDJWMCIsInkiOiJfS2N5TGo5dldNcHRubUt0bTQ2R3FE
ejh3Zjc0STVMS2dybDJHekgzblNFIn0sInByZXNlbnRhdGlvbl9qd2siOnsiY3J2IjoiU
C0yNTYiLCJrdHkiOiJFQyIsIngiOiJvQjFUUHJFX1FKSUw2MWZVT09LNURwS2dkOGoyem
JaSnRxcElMRFRKWDZJIiwieSI6IjNKcW5ya3VjTG9ia2RSdU9xWlhPUDlNTWxiRnllbkZ
PTHlHbEctRlBBQ00ifSwiYWxnIjoiU1UtRVMyNTYifQ
```

Figure 7: jwp-issuer-header-base64

Each payload must also be individually encoded:

The first payload is the JSON string "Doe" with the octet sequence of [ 34, 68, 111, 101, 34 ] and base64url-encoded as IkRvZSI.

The second payload is the JSON string "Jay" with the octet sequence of [ 34, 74, 97, 121, 34 ] and base64url-encoded as IkpheSI.

The third payload is the JSON string "jaydoe@example.org" with the octet sequence of [ 34, 106, 97, 121, 100, 111, 101, 64, 101, 120, 97, 109, 112, 108, 101, 46, 111, 114, 103, 34 ] and base64url-encoded as ImpheWRvZUBleGFtcGxlLm9yZyI.

The fourth payload is the JSON number 42 with the octet sequence of [52, 50] and base64url-encoded as NDI.

The Single Use algorithm utilizes multiple individual JWS Signatures. Each signature value is generated by creating a JWS with a single Protected Header with the associated alg value. In this example, the fixed header used for each JWS is the serialized JSON Object {"alg":"ES256"}. The JWS payload for each varies and the resulting signature value is used in its unencoded form (the octet string, not the base64url-encoded form).

The first signature is generated by creating a JWS using the fixed header with the payload set to the octet string of the JPT protected header from earlier. The resulting JWS signature using the Signer's *stable key* is the octet string of:

```
[44, 147, 34, 55, 167, 26, 18, 164, 161, 48, 158, 99, 60, 219, 108,
 240, 231, 172, 114, 163, 106, 230, 242, 146, 170, 71, 192, 117, 43,
 134, 36, 13, 70, 200, 156, 178, 173, 248, 228, 164, 1, 180, 112, 50,
 74, 102, 133, 32, 162, 97, 238, 228, 204, 216, 133, 53, 44, 107, 70,
 17, 93, 80, 103, 48]
```

Figure 8: jwp-issuer-header-signature

This process is repeated for the JPT payloads, using their octet strings as the payload in the ephemeral JWS in order to generate a signature using the *ephemeral key* for each:

The first payload signature is:

```
[171, 17, 93, 97, 129, 118, 193, 36, 150, 14, 229, 113, 60, 60, 114,
 243, 240, 152, 229, 218, 124, 218, 120, 150, 103, 43, 110, 177, 204,
 182, 28, 156, 72, 243, 36, 140, 160, 218, 241, 207, 27, 106, 88,
 133, 72, 43, 12, 143, 224, 43, 119, 76, 96, 216, 245, 111, 233, 39,
 131, 244, 158, 53, 210, 69]
```

Figure 9: jwp-payload-0-signature

The second payload signature is:

```
[112, 121, 108, 227, 203, 18, 91, 27, 206, 137, 237, 143, 12, 14,
 221, 135, 245, 254, 97, 132, 114, 48, 153, 34, 240, 93, 140, 194,
 108, 61, 251, 90, 107, 212, 17, 13, 191, 235, 8, 96, 1, 128, 121,
 186, 4, 144, 55, 112, 99, 92, 75, 226, 8, 15, 255, 85, 125, 229,
 110, 17, 245, 69, 87, 54]
```

Figure 10: jwp-payload-1-signature

The third payload signature is:

```
[195, 89, 195, 251, 210, 23, 69, 91, 7, 66, 9, 11, 213, 97, 77, 145,
 134, 185, 227, 131, 55, 23, 175, 179, 151, 206, 164, 26, 240, 254,
 25, 102, 110, 215, 202, 193, 166, 80, 58, 239, 217, 242, 167, 58,
 167, 134, 135, 44, 199, 142, 161, 2, 27, 222, 34, 12, 211, 107, 94,
 51, 190, 187, 120, 123]
```

Figure 11: jwp-payload-2-signature

The fourth payload signature is:

```
[236, 70, 36, 68, 119, 81, 128, 100, 48, 88, 219, 110, 19, 18, 179,
 236, 233, 176, 31, 202, 22, 139, 58, 101, 18, 216, 214, 39, 149,
 136, 148, 154, 94, 123, 191, 96, 67, 209, 211, 107, 100, 8, 57, 63,
 91, 80, 59, 227, 142, 73, 14, 250, 50, 191, 206, 224, 227, 103, 8,
 121, 50, 74, 220, 105]
```

Figure 12: jwp-payload-3-signature

Each payload's individual signature is concatenated in order, resulting in a larger octet string with a length of an individual signature (64 octets for ES256) multiplied by the number of payloads (4 for this example). These payload ephemeral signatures are then appended to the initial protected header stable signature. Using the above examples, the resulting octet string is 320 bytes in length (5 * 64):

```
[44, 147, 34, 55, 167, 26, 18, 164, 161, 48, 158, 99, 60, 219, 108,
 240, 231, 172, 114, 163, 106, 230, 242, 146, 170, 71, 192, 117, 43,
 134, 36, 13, 70, 200, 156, 178, 173, 248, 228, 164, 1, 180, 112, 50,
 74, 102, 133, 32, 162, 97, 238, 228, 204, 216, 133, 53, 44, 107, 70,
 17, 93, 80, 103, 48, 171, 17, 93, 97, 129, 118, 193, 36, 150, 14,
 229, 113, 60, 60, 114, 243, 240, 152, 229, 218, 124, 218, 120, 150,
 103, 43, 110, 177, 204, 182, 28, 156, 72, 243, 36, 140, 160, 218,
 241, 207, 27, 106, 88, 133, 72, 43, 12, 143, 224, 43, 119, 76, 96,
 216, 245, 111, 233, 39, 131, 244, 158, 53, 210, 69, 112, 121, 108,
 227, 203, 18, 91, 27, 206, 137, 237, 143, 12, 14, 221, 135, 245,
 254, 97, 132, 114, 48, 153, 34, 240, 93, 140, 194, 108, 61, 251, 90,
 107, 212, 17, 13, 191, 235, 8, 96, 1, 128, 121, 186, 4, 144, 55,
 112, 99, 92, 75, 226, 8, 15, 255, 85, 125, 229, 110, 17, 245, 69,
 87, 54, 195, 89, 195, 251, 210, 23, 69, 91, 7, 66, 9, 11, 213, 97,
 77, 145, 134, 185, 227, 131, 55, 23, 175, 179, 151, 206, 164, 26,
 240, 254, 25, 102, 110, 215, 202, 193, 166, 80, 58, 239, 217, 242,
 167, 58, 167, 134, 135, 44, 199, 142, 161, 2, 27, 222, 34, 12, 211,
 107, 94, 51, 190, 187, 120, 123, 236, 70, 36, 68, 119, 81, 128, 100,
 48, 88, 219, 110, 19, 18, 179, 236, 233, 176, 31, 202, 22, 139, 58,
 101, 18, 216, 214, 39, 149, 136, 148, 154, 94, 123, 191, 96, 67,
 209, 211, 107, 100, 8, 57, 63, 91, 80, 59, 227, 142, 73, 14, 250,
 50, 191, 206, 224, 227, 103, 8, 121, 50, 74, 220, 105]
```

Figure 13: jwp-signatures

The final Proof value from the Signer is the concatenated array of the header signature followed by all of the payload signatures, then base64url encoded.

The resulting JSON serialized JPT using the above examples is:

```
{
  "payloads": [
    "IkRvZSI",
    "IkpheSI",
    "ImpheWRvZUBleGFtcGxlLm9yZyI",
    "NDI"
  ],
  "issuer": [
    "eyJpc3MiOiJodHRwczovL2lzc3Vlci50bGQiLCJjbGFpbXMiOlsiZmFtaWx5X25h
    bWUiLCJnaXZlbl9uYW1lIiwiZW1haWwiLCJhZ2UiXSwidHlwIjoiSlBUIiwicHJvb
    2ZfandrIjp7ImNydiI6IlAtMjU2Iiwia3R5IjoiRUMiLCJ4IjoiYWNiSVFpdU1zM2
    k4X3VzekVqSjJ0cFR0Uk00RVUzeXo5MVBINkNkSDJWMCIsInkiOiJfS2N5TGo5dld
    NcHRubUt0bTQ2R3FEejh3Zjc0STVMS2dybDJHekgzblNFIn0sInByZXNlbnRhdGlv
    bl9qd2siOnsiY3J2IjoiUC0yNTYiLCJrdHkiOiJFQyIsIngiOiJvQjFUUHJFX1FKS
    Uw2MWZVT09LNUR wS2dkOGoyemJaSnRxcElMRFRKWDZJIiwieSI6IjNKcW5ya3VjT
    G9ia2RSdU9xWlhPUDlNTWxiRnllbkZPTHlHbEctRlBBQ00ifSwiYWxnIjoiU1UtRV
    MyNTYifQ"
  ],
  "proof": [
    "LJMiN6caEqShMJ5jPNts8OescqNq5vKSqkfAdSuGJA1GyJyyrfjkpAG0cDJKZoUg
    omHu5MzYhTUsa0YRXVBnMKsRXWGBdsEklg7lcTw8cvPwmOXafNp4lmcrbrHMthycS
    PMkjKDa8c8baliFSCsMj-Ard0xg2PVv6SeD9J410kVweWzjyxJbG86J7Y8MDt2H9f
    5hhHIwmSLwXYzCbD37WmvUEQ2_6whgAYB5ugSQN3BjXEviCA__VX3lbhH1RVc2w1n
    D-9IXRVsHQgkL1WFNkYa544M3F6-zl86kGvD-GWZu18rBplA679nypzqnhocsx46h
    AhveIgzTa14zvrt4e-xGJER3UYBkMFjbbhMSs-zpsB_KFos6ZRLY1ieViJSaXnu_Y
    EPR02tkCDk_W1A7445JDvoyv87g42cIeTJK3Gk"
  ]
}
```


Figure 14: jwp-final

The compact serialization of the same JPT is:

```
eyJpc3MiOiJodHRwczovL2lzc3Vlci50bGQiLCJjbGFpbXMiOlsiZmFtaWx5X25hbWUiL
CJnaXZlbl9uYW1lIiwiZW1haWwiLCJhZ2UiXSwidHlwIjoiSlBUIiwicHJvb2ZfandrIj
p7ImNydiI6IlAtMjU2Iiwia3R5IjoiRUMiLCJ4IjoiYWNiSVFpdU1zM2k4X3VzekVqSjJ
0cFR0Uk00RVUzeXo5MVBINkNkSDJWMCIsInkiOiJfS2N5TGo5dldNcHRubUt0bTQ2R3FE
ejh3Zjc0STVMS2dybDJHekgzblNFIn0sInByZXNlbnRhdGlvbl9qd2siOnsiY3J2IjoiU
C0yNTYiLCJrdHkiOiJFQyIsIngiOiJvQjFUUHJFX1FKSUw2MWZVT09LNURwS2dkOGoyem
JaSnRxcElMRFRKWDZJIiwieSI6IjNKcW5ya3VjTG9ia2RSdU9xWlhPUDlNTWxiRnllbkZ
PTHlHbEctRlBBQ00ifSwiYWxnIjoiU1UtRVMyNTYifQ.IkRvZSI~IkpheSI~ImpheWRvZ
UBleGFtcGxlLm9yZyI~NDI.LJMiN6caEqShMJ5jPNts8OescqNq5vKSqkfAdSuGJA1GyJ
yyrfjkpAG0cDJKZoUgomHu5MzYhTUsa0YRXVBnMKsRXWGBdsEklg7lcTw8cvPwmOXafNp
4lmcrbrHMthycSPMkjKDa8c8baliFSCsMj-Ard0xg2PVv6SeD9J410kVweWzjyxJbG86J
7Y8MDt2H9f5hhHIwmSLwXYzCbD37WmvUEQ2_6whgAYB5ugSQN3BjXEviCA__VX3lbhH1R
Vc2w1nD-9IXRVsHQgkL1WFNkYa544M3F6-zl86kGvD-GWZu18rBplA679nypzqnhocsx4
6hAhveIgzTa14zvrt4e-xGJER3UYBkMFjbbhMSs-zpsB_KFos6ZRLY1ieViJSaXnu_YEP
R02tkCDk_W1A7445JDvoyv87g42cIeTJK3Gk
```

Figure 15: jwp-compact

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

```
{
  "nonce": "uTEB371l1pzWJl7afB0wi0HWUNk1Le-bComFLxa8K-s"
}
```

Figure 16: jwp-presentation-header

When serialized, this results in the following octets:

```
[123, 34, 110, 111, 110, 99, 101, 34, 58, 34, 117, 84, 69, 66, 51,
 55, 49, 108, 49, 112, 122, 87, 74, 108, 55, 97, 102, 66, 48, 119,
 105, 48, 72, 87, 85, 78, 107, 49, 76, 101, 45, 98, 67, 111, 109, 70,
 76, 120, 97, 56, 75, 45, 115, 34, 125]
```

Figure 17: jwp-presentation-header-octets

And when base64url encoded results in the string:

```
eyJub25jZSI6InVURUIzNzFsMXB6V0psN2FmQjB3aTBIV1VOazFMZS1iQ29tRkx4YThLL
XMifQ
```

Figure 18: jwp-presentation-header-base64

When signed with the holder's presentation key, the resulting signature octets are:

```
[31, 117, 70, 58, 39, 174, 117, 149, 177, 169, 31, 94, 215, 230, 218,
 30, 224, 31, 16, 63, 240, 109, 112, 144, 29, 61, 199, 79, 100, 162,
 125, 158, 43, 216, 19, 191, 136, 138, 195, 129, 143, 233, 24, 116,
 216, 241, 8, 1, 80, 47, 158, 27, 110, 137, 40, 76, 156, 250, 149,
 195, 91, 66, 5, 216]
```

Figure 19: jwp-presentation-header-signature

Then by applying selective disclosure of only the given name and age claims (family name and email hidden), the proof value including the signature of the presentation header and removing the ephemeral signatures of the family name and email payloads results in the following octet array:

```
[44, 147, 34, 55, 167, 26, 18, 164, 161, 48, 158, 99, 60, 219, 108,
 240, 231, 172, 114, 163, 106, 230, 242, 146, 170, 71, 192, 117, 43,
 134, 36, 13, 70, 200, 156, 178, 173, 248, 228, 164, 1, 180, 112, 50,
 74, 102, 133, 32, 162, 97, 238, 228, 204, 216, 133, 53, 44, 107, 70,
 17, 93, 80, 103, 48, 31, 117, 70, 58, 39, 174, 117, 149, 177, 169,
 31, 94, 215, 230, 218, 30, 224, 31, 16, 63, 240, 109, 112, 144, 29,
 61, 199, 79, 100, 162, 125, 158, 43, 216, 19, 191, 136, 138, 195,
 129, 143, 233, 24, 116, 216, 241, 8, 1, 80, 47, 158, 27, 110, 137,
 40, 76, 156, 250, 149, 195, 91, 66, 5, 216, 112, 121, 108, 227, 203,
 18, 91, 27, 206, 137, 237, 143, 12, 14, 221, 135, 245, 254, 97, 132,
 114, 48, 153, 34, 240, 93, 140, 194, 108, 61, 251, 90, 107, 212, 17,
 13, 191, 235, 8, 96, 1, 128, 121, 186, 4, 144, 55, 112, 99, 92, 75,
 226, 8, 15, 255, 85, 125, 229, 110, 17, 245, 69, 87, 54, 236, 70,
 36, 68, 119, 81, 128, 100, 48, 88, 219, 110, 19, 18, 179, 236, 233,
 176, 31, 202, 22, 139, 58, 101, 18, 216, 214, 39, 149, 136, 148,
 154, 94, 123, 191, 96, 67, 209, 211, 107, 100, 8, 57, 63, 91, 80,
 59, 227, 142, 73, 14, 250, 50, 191, 206, 224, 227, 103, 8, 121, 50,
 74, 220, 105]
```

Figure 20: jwp-presentation-signatures

The resulting presented JPT in JSON serialization:

```
{
  "payloads": [
    null,
    "IkpheSI",
    null,
    "NDI"
  ],
  "issuer": "eyJpc3MiOiJodHRwczovL2lzc3Vlci50bGQiLCJjbGFpbXMiOlsiZmFt
  aWx5X25hbWUiLCJnaXZlbl9uYW1lIiwiZW1haWwiLCJhZ2UiXSwidHlwIjoiSlBUIiw
  icHJvb2ZfandrIjp7ImNydiI6IlAtMjU2Iiwia3R5IjoiRUMiLCJ4IjoiYWNiSVFpdU
  1zM2k4X3VzekVqSjJ0cFR0Uk00RVUzeXo5MVBINkNkSDJWMCIsInkiOiJfS2N5TGo5d
  ldNcHRubUt0bTQ2R3FEejh3Zjc0STVMS2dybDJHekgzblNFIn0sInByZXNlbnRhdGlv
  bl9qd2siOnsiY3J2IjoiUC0yNTYiLCJrdHkiOiJFQyIsIngiOiJvQjFUUHJFX1FKSUw
  2MWZVT09LNURwS2dkOGoyemJaSnRxcElMRFRKWDZJIiwieSI6IjNKcW5ya3VjTG9ia2
  RSdU9xWlhPUDlNTWxiRnllbkZPTHlHbEctRlBBQ00ifSwiYWxnIjoiU1UtRVMyNTYif
  Q",
  "proof": [
    "LJMiN6caEqShMJ5jPNts8OescqNq5vKSqkfAdSuGJA1GyJyyrfjkpAG0cDJKZoUg
    omHu5MzYhTUsa0YRXVBnMB91RjonrnWVsakfXtfm2h7gHxA_8G1wkB09x09kon2eK
    9gTv4iKw4GP6Rh02PEIAVAvnhtuiShMnPqVw1tCBdhweWzjyxJbG86J7Y8MDt2H9f
    5hhHIwmSLwXYzCbD37WmvUEQ2_6whgAYB5ugSQN3BjXEviCA__VX3lbhH1RVc27EY
    kRHdRgGQwWNtuExKz7OmwH8oWizplEtjWJ5WIlJpee79gQ9HTa2QIOT9bUDvjjkkO
    -jK_zuDjZwh5MkrcaQ"
  ],
  "presentation": "eyJub25jZSI6InVURUIzNzFsMXB6V0psN2FmQjB3aTBIV1VOaz
  FMZS1iQ29tRkx4YThLLXMifQ"
  ]
}
```


Figure 21: jwp-final-presentation

And also in compact serialization:

eyJpc3MiOiJodHRwczovL2lzc3Vlci50bGQiLCJjbGFpbXMiOlsiZmFtaWx5X25hbWUiL
CJnaXZlbl9uYW1lIiwiZW1haWwiLCJhZ2UiXSwidHlwIjoiSlBUIiwicHJvb2ZfandrIj
p7ImNydiI6IlAtMjU2Iiwia3R5IjoiRUMiLCJ4IjoiYWNiSVFpdU1zM2k4X3VzekVqSjJ
0cFR0Uk00RVUzeXo5MVBINkNkSDJWMCIsInkiOiJfS2N5TGo5dldNcHRubUt0bTQ2R3FE
ejh3Zjc0STVMS2dybDJHekgzblNFIn0sInByZXNlbnRhdGlvbl9qd2siOnsiY3J2IjoiU
C0yNTYiLCJrdHkiOiJFQyIsIngiOiJvQjFUUHJFX1FKSUw2MWZVT09LNURwS2dkOGoyem
JaSnRxcElMRFRKWDZJIiwieSI6IjNKcW5ya3VjTG9ia2RSdU9xWlhPUDlNTWxiRnllbkZ
PTHlHbEctRlBBQ00ifSwiYWxnIjoiU1UtRVMyNTYifQ.eyJub25jZSI6InVURUIzNzFsM
XB6V0psN2FmQjB3aTBIV1VOazFMZS1iQ29tRkx4YThLLXMifQ.~IkpheSI~~NDI.LJMiN
6caEqShMJ5jPNts8OescqNq5vKSqkfAdSuGJA1GyJyyrfjkpAG0cDJKZoUgomHu5MzYhT
Usa0YRXVBnMB91RjonrnWVsakfXtfm2h7gHxA_8G1wkB09x09kon2eK9gTv4iKw4GP6Rh
02PEIAVAvnhtuiShMnPqVw1tCBdhweWzjyxJbG86J7Y8MDt2H9f5hhHIwmSLwXYzCbD37
WmvUEQ2_6whgAYB5ugSQN3BjXEviCA__VX3lbhH1RVc27EYkRHdRgGQwWNtuExKz7OmwH
8oWizplEtjWJ5WIlJpee79gQ9HTa2QIOT9bUDvjjkkO-jK_zuDjZwh5MkrcaQ

Figure 22: jwp-compact-presentation

### A.2. Example Multi-Use JWP

See JPA BBS-DRAFT-5 example.

## Appendix B. Acknowledgements

This work was incubated in the DIF Applied Cryptography Working Group.

We would like to thank Brent Zundel for his valuable contributions to this specification.

## Appendix C. Registries

- Issuer Protected Header
- Presentation Protected Header

## Appendix D. Document History

[[ To be removed from the final specification ]]

-03

- Improvements resulting from a full proofreading.
- Populated IANA Considerations section.
- Specified JWP Header Parameters.
- Specified representation of zero-length disclosed payloads for the compact serialization.
- Specified that algorithms may supply multiple octet strings for the proof, which are separated by ~ characters in the compact serialization.
- Updated to use BBS draft -05.

*Added Terminology Section.

-02

*Update reference to current BBS algorithm

-01

*Correct cross-references within group.

-00

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

## Authors' Addresses


Jeremie Miller
Ping Identity

Email: jmiller@pingidentity.com

David Waite
Ping Identity

Email: dwaite+jwp@pingidentity.com

Michael B. Jones
Self-Issued Consulting

Email: michael_b_jones@hotmail.com
URI: https://self-issued.info/