Workgroup: jose
Internet-Draft:
draft-jmiller-jose-json-web-proof-01
Published: 10 March 2023
Intended Status: Standards Track
Expires: 11 September 2023
Authors: J. Miller
Ping Identity
D. Waite
Ping Identity
M. Jones
Microsoft

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 and support binding mechanisms.

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+

* 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 Verifiable Credentials Data Model v1.1. The term "presentation" is also used as defined by this source, but the term "credential" is avoided in this specification in order to minimize confusion with other definitions.

2.1. Abbreviations

*ZKP: Zero-Knowledge Proof

*JWP: JSON Web Proof (this specification)

*JPA: JSON Proof Algorithms

*JPT: 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 singular 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.

In order to fully support the newer privacy primitives, JWP introduces the three roles of issuer, holder, and verifier as defined by the VC Data Model. 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.

A JWP is initially created by the issuer using the issue interaction with an implementation. 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 confirm to check the integrity protection of the header and all payloads using the given proof value.

After validation, the holder uses present 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 verify 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. The unlinkability of the resulting presented JWP is only provided when supported by the underlying algorithm.

Algorithm definitions that support JWPs are being done in separate companion specifications - just as the JSON Web Algorithms [RFC7518] specification does for JWS and JWE [RFC7516]. The JSON Proof Algorithms specification defines how an initial set of algorithms are used with JWP.

4. JWP Forms

A JWP is always in one of two forms: the issued form or the presented form. The significant 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 a single octet string that is only 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.

4.1. 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.

4.1.1. 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.

Every issuer protected header MUST have, at minimum, an alg value that identifies a valid JPA.

For example:

```
{
    "alg":"BBS-X"
}
```

4.1.2. 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.

4.1.3. Issuer Proof

The proof value is a binary octet string that is 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 only for the holder to perform validation, checking that the issuer header and all payloads are properly encoded and protected by the given proof.

4.2. 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, 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.

4.2.1. 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.

4.2.2. 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.

4.2.3. 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.

5. 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.

5.1. 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.

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.

5.2. JSON Serialization

Non-disclosed payloads in the JSON serialization are represented with a null value in the payloads array.

Example flattened JSON serialization showing the presentation form with both the issuer and presentation headers along with the first and third payloads hidden.

```
{
  "payloads": [
    null,
    "IkpheSI",
    null,
    "NDI"
  ],
  "issuer": "eyJpc3MiOiJodHRwczovL2lzc3Vlci50bGQiLCJjbGFpbXMiOlsiZmFt
  aWx5X25hbWUiLCJnaXZlbl9uYW1lIiwiZW1haWwiLCJhZ2UiXSwidHlwIjoiSlBUIiw
  icHJvb2ZfandrIjp7ImNydiI6IlAtMjU2Iiwia3R5IjoiRUMiLCJ4IjoiYWNiSVFpdU
  1zM2k4X3VzekVqSjJ0cFR0Uk00RVUzeXo5MVBINkNkSDJWMCIsInkiOiJfS2N5TGo5d
  ldNcHRubUt0bTQ2R3FEejh3Zjc0STVMS2dybDJHekgzblNFIn0sInByZXNlbnRhdGlv
  bl9qd2siOnsiY3J2IjoiUC0yNTYiLCJrdHkiOiJFQyIsIngiOiJvQjFUUHJFX1FKSUw
  2MWZVT09LNURwS2dkOGoyemJaSnRxcElMRFRKWDZJIiwieSI6IjNKcW5ya3VjTG9ia2
  RSdU9xWlhPUDlNTWxiRnllbkZPTHlHbEctRlBBQ00ifSwiYWxnIjoiU1UtRVMyNTYif
  Q",
  "proof": "LJMiN6caEqShMJ5jPNts8OescqNq5vKSqkfAdSuGJA1GyJyyrfjkpAG0c
  DJKZoUgomHu5MzYhTUsa0YRXVBnMB91RjonrnWVsakfXtfm2h7gHxA_8G1wkB09x09k
  on2eK9gTv4iKw4GP6Rh02PEIAVAvnhtuiShMnPqVw1tCBdhweWzjyxJbG86J7Y8MDt2
  H9f5hhHIwmSLwXYzCbD37WmvUEQ2_6whgAYB5ugSQN3BjXEviCA__VX3lbhH1RVc27E
  YkRHdRgGQwWNtuExKz7OmwH8oWizplEtjWJ5WIlJpee79gQ9HTa2QIOT9bUDvjjkkO-
  jK_zuDjZwh5MkrcaQ",
  "presentation": "eyJub25jZSI6InVURUIzNzFsMXB6V0psN2FmQjB3aTBIV1VOaz
  FMZS1iQ29tRkx4YThLLXMifQ"
}
```


Figure 1: jwp-final-presentation

6. Security Considerations

Notes to be expanded:

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

7. IANA Considerations

This document has no IANA actions.

8. Informative References

[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>.

[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>.

[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>.

[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>.

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

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

The third payload is the 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 string 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 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": "eyJpc3MiOiJodHRwczovL2lzc3Vlci50bGQiLCJjbGFpbXMiOlsiZmFt
  aWx5X25hbWUiLCJnaXZlbl9uYW1lIiwiZW1haWwiLCJhZ2UiXSwidHlwIjoiSlBUIiw
  icHJvb2ZfandrIjp7ImNydiI6IlAtMjU2Iiwia3R5IjoiRUMiLCJ4IjoiYWNiSVFpdU
  1zM2k4X3VzekVqSjJ0cFR0Uk00RVUzeXo5MVBINkNkSDJWMCIsInkiOiJfS2N5TGo5d
  ldNcHRubUt0bTQ2R3FEejh3Zjc0STVMS2dybDJHekgzblNFIn0sInByZXNlbnRhdGlv
  bl9qd2siOnsiY3J2IjoiUC0yNTYiLCJrdHkiOiJFQyIsIngiOiJvQjFUUHJFX1FKSUw
  2MWZVT09LNUR wS2dkOGoyemJaSnRxcElMRFRKWDZJIiwieSI6IjNKcW5ya3VjTG9ia
  2RSdU9xWlhPUDlN TWxiRnllbkZPTHlHbEctRlBBQ00ifSwiYWxnIjoiU1UtRVMyNTY
  ifQ",
  "proof": "LJMiN6caEqShMJ5jPNts8OescqNq5vKSqkfAdSuGJA1GyJyyrfjkpAG0c
  DJKZoUgomHu5MzYhTUsa0YRXVBnMKsRXWGBdsEklg7lcTw8cvPwmOXafNp4lmcrbrHM
  thycSPMkjKDa8c8baliFSCsMj-Ard0xg2PVv6SeD9J410kVweWzjyxJbG86J7Y8MDt2
  H9f5hhHIwmSLwXYzCbD37WmvUEQ2_6whgAYB5ugSQN3BjXEviCA__VX3lbhH1RVc2w1
  nD-9IXRVsHQgkL1WFNkYa544M3F6-zl86kGvD-GWZu18rBplA679nypzqnhocsx46hA
  hveIgzTa14zvrt4e-xGJER3UYBkMFjbbhMSs-zpsB_KFos6ZRLY1ieViJSaXnu_YEPR
  02tkCDk_W1A7 445JDvoyv87g42cIeTJK3Gk"
}
```


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": "LJMiN6caEqShMJ5jPNts8OescqNq5vKSqkfAdSuGJA1GyJyyrfjkpAG0c
  DJKZoUgomHu5MzYhTUsa0YRXVBnMB91RjonrnWVsakfXtfm2h7gHxA_8G1wkB09x09k
  on2eK9gTv4iKw4GP6Rh02PEIAVAvnhtuiShMnPqVw1tCBdhweWzjyxJbG86J7Y8MDt2
  H9f5hhHIwmSLwXYzCbD37WmvUEQ2_6whgAYB5ugSQN3BjXEviCA__VX3lbhH1RVc27E
  YkRHdRgGQwWNtuExKz7OmwH8oWizplEtjWJ5WIlJpee79gQ9HTa2QIOT9bUDvjjkkO-
  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-X example.

Appendix B. Acknowledgements

TBD

Appendix C. Registries

- *Issuer Protected Header
- *Presentation Protected Header

Appendix D. Document History

[[To be removed from the final specification]]

-01

- *Applied editorial improvements

-00

- *First individual draft targeting JOSE working group

Authors' Addresses

Jeremie Miller
Ping Identity

Email: jmiller@pingidentity.com

David Waite
Ping Identity

Email: dwaite+jwp@pingidentity.com

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: https://self-issued.info/