Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 May 2026

C. Wendt
D. Hancock
Somos Inc.

24 November 2025

JWTClaimConstraints profile of ACME Authority Token
draft-ietf-acme-authority-token-jwtclaimcon-00

Abstract

This document defines an authority token profile for the validation of JWTClaimConstraints and EnhancedJWTClaimConstraints certificate extensions within the Automated Certificate Management Environment (ACME) protocol. This profile is based on the Authority Token framework and establishes the specific ACME identifier type, challenge mechanism, and token format necessary to authorize a client to request a certificate containing these constraints.

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 28 May 2026.

Copyright Notice

Table of Contents

1. Introduction

The validation of certificate extensions that constrain the use of credentials, such as JWTClaimConstraints defined in [RFC8226] and EnhancedJWTClaimConstraints defined in [RFC9118], is critical for defining the scope of an issued certificate. This document specifies an authority token profile for validating these constraints, modeled after the authority token framework established in and the TNAuthList validation defined in.

This profile facilitates proper delegation and authorization for entities requesting certificates under ACME and similar frameworks. It defines the use of the JWTClaimConstraints Authority Token in the ACME challenge to prove an authoritative or trusted use of the contents of the JWTClaimConstraints based on the issuer of the token.

This document also discusses the ability for an authority to authorize the creation of CA types of certificates for delegation as defined in [RFC9060].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. ACME new-order Identifiers for JWTClaimConstraints

In [RFC8555], Section 7 defines the procedure that an ACME client uses to order a new certificate from a Certification Authority (CA). This draft defines a new type of identifier object called JWTClaimConstraints. A JWTClaimConstraints identifier contains the Token Claim Constraints information to be populated in the JWTClaimConstraints or EnhancedJWTClaimConstraints of the new certificate.

For the JWTClaimConstraints identifier, the new-order request includes a type set to the string "JWTClaimConstraints". The value of the JWTClaimConstraints identifier MUST be set to the details of the JWTClaimConstraints requested.

The format of the string that represents the JWTClaimConstraints MUST be constructed using base64url encoding, as per [RFC8555] base64url encoding described in Section 5 of [RFC4648] according to the profile specified in JSON Web Signature in Section 2 of [RFC7515]. The base64url encoding MUST NOT include any padding characters and the JWTClaimConstraints ASN.1 object MUST be encoded using DER encoding rules.

An example of an ACME order object "identifiers" field containing a JWTClaimConstraints certificate:

```
 "identifiers": [{"type":"JWTClaimConstraints",
   "value":"F83n2a...avn27DN3"}]
```

where the "value" object string represents the arbitrary length base64url encoded string.

A full new-order request would look as follows,

```
POST /acme/new-order HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/evOfKhNU60wg",
    "nonce": "5XJ1L3lEkMG7tR6pA00clA",
    "url": "https://example.com/acme/new-order"
  }),
  "payload": base64url({
    "identifiers": [{"type":"JWTClaimConstraints",
      "value":"F83n...n27DN3"}],
    "notBefore": "2025-01-01T00:00:00Z",
    "notAfter": "2025-01-08T00:00:00Z"
  }),
  "signature": "H6ZXtGjTZyUnPeKn...wEA4TklBdh3e454g"
}
```

On receiving a valid new-order request, the ACME server creates an authorization object, [RFC8555] Section 7.1.4, containing the challenge that the ACME client must satisfy to demonstrate authority for the identifiers specified by the new order (in this case, the JWTClaimConstraints identifier). The CA adds the authorization object URL to the "authorizations" field of the order object, and returns the order object to the ACME client in the body of a 201 (Created) response.

```
HTTP/1.1 201 Created
Content-Type: application/json
Replay-Nonce: MYAuvOpaoIiywTezizk5vw
Location: https://example.com/acme/order/1234

{
  "status": "pending",
  "expires": "2025-01-08T00:00:00Z",

  "notBefore": "2025-01-01T00:00:00Z",
  "notAfter": "2025-01-08T00:00:00Z",
  "identifiers":[{"type":"JWTClaimConstraints",
                  "value":"F83n2a...avn27DN3"}],

  "authorizations": [
   "https://example.com/acme/authz/1234"
  ],
  "finalize": "https://example.com/acme/order/1234/finalize"
}
```

4. JWTClaimConstraints Authorization

On receiving the new-order response, the ACME client queries the referenced authorization object to obtain the challenges for the identifier contained in the new-order request.

```
POST /acme/authz/1234 HTTP/1.1
    Host: example.com
    Content-Type: application/jose+json

    {
      "protected": base64url({
        "alg": "ES256",
        "kid": " https://example.com/acme/acct/evOfKhNU60wg",
        "nonce": "uQpSjlRb4vQVCjVYAyyUWg",
        "url": "https://example.com/acme/authz/1234"
      }),
      "payload": "",
      "signature": "nuSDISbWG8mMgE7H...QyVUL68yzf3Zawps"
    }
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Link: <https://example.com/acme/some-directory>;rel="index"

{
  "status": "pending",
  "expires": "2025-01-08T00:00:00Z",

  "identifier": {
    "type:"JWTClaimConstraints",
    "value":"F83n2a...avn27DN3"
  },

  "challenges": [
    {
      "type": "tkauth-01",
      "tkauth-type": "atc",
      "token-authority": "https://authority.example.org",
      "url": "https://boulder.example.com/acme/chall/prV_B7yEyA4",
      "token": "IlirfxKKXAsHtmzK29Pj8A"
    }
  ]
}
```

When processing a certificate order containing an identifier of type "JWTClaimConstraints", a CA uses the Authority Token challenge type of "tkauth-01" with a "tkauth-type" of "atc", defined in [RFC9447], to verify that the requesting ACME client has authenticated and authorized control over the requested resources represented by the "JWTClaimConstraints" value.

The challenge "token-authority" parameter is OPTIONAL. If a "token-authority" parameter is present, the ACME client MAY use this value to identify the URL representing the Token Authority that will provide the JWTClaimConstraints Authority Token response to the challenge. If the "token-authority" parameter is not present, the ACME client MUST identify the Token Authority based on locally configured information or local policies.

The ACME client responds to the challenge by posting the JWTClaimConstraints Authority Token to the challenge URL identified in the returned ACME authorization object.

```
POST /acme/chall/prV_B7yEyA4 HTTP/1.1
Host: boulder.example.com
Content-Type: application/jose+json

{
  "protected": base64url({
  "alg": "ES256",
  "kid": "https://example.com/acme/acct/evOfKhNU60wg",
  "nonce": "Q_s3MWoqT05TrdkM2MTDcw",
  "url": "https://boulder.example.com/acme/authz/asdf/0"
  }),
  "payload": base64url({
  "tkauth": "DGyRejmCefe7v4N...vb29HhjjLPSggwiE"
  }),
  "signature": "9cbg5JO1Gf5YLjjz...SpkUfcdPai9uVYYQ"
}
```

The "tkauth" field in the challenge response object is specific to the tkauth-01 challenge type. When responding to a challenge for a JWTClaimConstraints identifier, this field SHALL contain the JWTClaimConstraints Authority Token defined in the next section.

5. JWTClaimConstraints Authority Token

The JWTClaimConstraints Authority Token is a profile instance of the ACME Authority Token defined in [RFC9447].

The JWTClaimConstraints Authority Token Protected header MUST comply with the Authority Token Protected header as defined in [RFC9447].

5.1. JWTClaimConstraints Authority Token Payload

The JWTClaimConstraints Authority Token Payload MUST include the mandatory claims "exp", "jti", and "atc", and MAY include the optional claims defined for the Authority Token.

5.1.1. "iss" claim

The "iss" claim is an optional claim defined in [RFC7519] Section 4.1.1. It can be used as a URL identifying the Token Authority that issued the JWTClaimConstraints Authority Token beyond the "x5u" or other Header claims that identify the location of the certificate or certificate chain of the Token Authority used to validate the JWTClaimConstraints Authority Token.

5.1.2. "exp" claim

The "exp" claim, defined in [RFC7519] Section 4.1.4, MUST be included and contains the DateTime value of the ending date and time that the JWTClaimConstraints Authority Token expires.

5.1.3. "jti" claim

The "jti" claim, defined in [RFC7519] Section 4.1.7, MUST be included and contains a unique identifier for this JWTClaimConstraints Authority Token transaction.

5.1.4. "atc" claim

The "atc" claim MUST be included and is defined in [RFC9447]. It contains a JSON object with the following elements:

* a "tktype" key with a string value equal to "JWTClaimConstraints" to represent a JWTClaimConstraints profile of the authority token [RFC9447] defined by this document. "tktype" is a required key and MUST be included.

* a "tkvalue" key with a string value equal to the base64url encoding of the JWTClaimConstraints or EnhancedJWTClaimConstraints certificate extension ASN.1 object using DER encoding rules. "tkvalue" is a required key and MUST be included.

* a "ca" key with a boolean value set to false (since the JWTClaimConstraints extension is applicable only to end-entity certificates). "ca" is an optional key; if not included, the "ca" value is considered false by default.

* a "fingerprint" key is constructed as defined in [RFC8555] Section 8.1 corresponding to the computation of the "Thumbprint" step using the ACME account key credentials. "fingerprint" is a required key and MUST be included.

An example of the JWTClaimConstraints Authority Token is as follows:

```
{
  "protected": base64url({
    "typ":"JWT",
    "alg":"ES256",
    "x5u":"https://authority.example.org/cert"
  }),
  "payload": base64url({
    "iss":"https://authority.example.org",
    "exp":1640995200,
    "jti":"id6098364921",
    "atc":{"tktype":"JWTClaimConstraints",
      "tkvalue":"F83n2a...avn27DN3",
      "ca":false,
      "fingerprint":"SHA256 56:3E:CF:AE:83:CA:4D:15:B0:29:FF:1B:71:
       D3:BA:B9:19:81:F8:50:9B:DF:4A:D4:39:72:E2:B1:F0:B9:38:E3"}
  }),
  "signature": "9cbg5JO1Gf5YLjjz...SpkUfcdPai9uVYYQ"
}
```

5.2. Acquiring the token from the Token Authority

Following [RFC9447] Section 5, the authority token should be acquired using a RESTful HTTP POST transaction as follows:

```
POST /at/account/:id/token HTTP/1.1
Host: authority.example.org
Content-Type: application/json
```

The request will pass the account id as a string in the request parameter "id". There is assumed to be a corresponding authentication procedure that can be verified for the success of this transaction, for example, an HTTP Authorization header field containing valid authorization credentials as defined in [RFC7231] Section 14.8.

The body of the POST request MUST contain a JSON object with key value pairs corresponding to values that are requested as the content of the claims in the issued token.

As an example, the body SHOULD contain a JSON object as follows:

```
{
  "atc": {
    "tktype":"JWTClaimConstraints",
    "tkvalue":"F83n2a...avn27DN3",
    "ca":false,
    "fingerprint":"SHA256 56:3E:CF:AE:83:CA:4D:15:B0:29:FF:1B:71:D3
      :BA:B9:19:81:F8:50:9B:DF:4A:D4:39:72:E2:B1:F0:B9:38:E3"
  }
}
```

The response to the POST request if successful returns a 200 OK with a JSON body that contains, at a minimum, the JWTClaimConstraints Authority Token as a JSON object with a key of "token" and the base64url encoded string representing the atc token. JSON is easily extensible, so users of this specification may want to pass other pieces of information relevant to a specific application.

An example successful response would be as follows:

```
HTTP/1.1 200 OK
Content-Type: application/json

{"token": "DGyRejmCefe7v4N...vb29HhjjLPSggwiE"}
```

If the request is not successful, the response should indicate the error condition. Specifically, for the case that the authorization credentials are invalid or if the Account ID provided does not exist, the response code MUST be 403 - Forbidden. Other 4xx and 5xx responses MUST follow standard [RFC7231] HTTP error condition conventions.

5.3. Token Authority Responsibilities

When creating the JWTClaimConstraints Authority Token, the Token Authority MUST validate that the information contained in the ASN.1 JWTClaimConstraints accurately represents the corresponding JWTClaimConstraint resources the requesting party is authorized to represent based on their pre-established and verified secure relationship between the Token Authority and the requesting party.

The fingerprint in the token request is not meant to be verified by the Token Authority. Rather, it is meant to be signed as part of the token so that the party that requests the token can, as part of the challenge response, allow the ACME server to validate that the token requested and used came from the same party that controls the ACME client.

5.4. Scope of the JWTClaimConstraints

Because this specification involves the JWTClaimConstraints and EnhancedJWTClaimConstraints extensions, the client MAY request an Authority Token with some subset of its own authority as the JWTClaimConstraints provided in the "tkvalue" element of the "atc" JSON object. JWTClaimConstraints can be constructed to define a limited scope of claims and claim values the client has authority over.

5.5. ACME Challenges requiring multiple Authority Tokens

The ACME new-order request may include multiple identifiers, each of which is authorized separately. With the introduction of this specification, for STIR certificates [RFC8226], a certificate order may require two Authority Token identifier types:

* The JWTClaimConstraints identifier defined in this document, and

* The TNAuthList identifier defined in [RFC9448].

Other Authority Token types may be introduced in future Authority Token profile specifications with similar requirements.

This section describes scenarios where a new-order request contains both of these identifier types. In such cases, the CA requires the ACME client to provide both a JWTClaimConstraints Authority Token and a TNAuthList Authority Token as part of the challenge response.

The TNAuthList Authority Token authorizes the token holder to obtain certificates containing a TNAuthList extension whose scope is less than or equal to the scope of the TNAuthList identifier in the token.

The JWTClaimConstraints Authority Token authorizes the token holder to obtain a certificate containing a JWTClaimConstraints or EnhancedJWTClaimConstraints extension, provided that the extension is within the scope of the JWTClaimConstraints identifier in the token. Since these two certificate extensions constrain the resources and claims available to the certificate, there is an inherent interaction between these two types of Authority Tokens.

5.5.1. Examples of ACME Challenges requiring two Authority Tokens

In the examples that follow, the requesting user is authorized to use a set of telephone numbers (TNs) and is authorized to assert various claims and claim values. To support these capabilities, the user obtains both a TNAuthList Authority Token and a JWTClaimConstraints Authority Token from the appropriate Token Authority. These two tokens MAY be issued by the same Token Authority or by distinct entities.

The examples illustrate how the two types of Authority Tokens are used. The TNAuthList Authority Token specifies the authorized TNs. The content of the JWTClaimConstraints Authority Token varies based on the user's authority to assert claims and values.

5.5.1.1. No Extended Claims Authorized

In the first case, the requesting user is authorized to use a set of TN(s), but no other optional claim information. Accordingly, the JWTClaimConstraints Authority Token contains an EnhancedJWTClaimConstraints extension that prohibits all optional claims relevant to the application (e.g., using a mustExclude constraint).

```
SEQUENCE {
  mustExclude [2] {
    SEQUENCE {
      IA5String 'attest'
      IA5String 'origid'
      IA5String 'div'
      IA5String 'rph'
      IA5String 'sph'
      IA5String 'rcd'
      IA5String 'rcdi'
      IA5String 'crn'
      }
    }
  }
```

A simpler alternative for users not authorized to include optional claims is to submit a new-order request containing only a TNAuthList identifier. In this case, the absence of a JWTClaimConstraints identifier MAY trigger local policy in the CA to include a restrictive EnhancedJWTClaimConstraints extension in the issued certificate.

5.5.1.2. Extended Claims Authorized (Uniform Constraints)

In the second case, the user is authorized to assert a specific set of claim information, and these constraints apply uniformly to all TN(s) the user is authorized to use. The corresponding JWTClaimConstraints Authority Token contains a JWTClaimConstraints or EnhancedJWTClaimConstraints extension that permits a specific set of claim values (e.g., using a permittedValues constraint) that applies across the user's authorized TNs.

```
SEQUENCE {
  permittedValues [1] {
    SEQUENCE {
      SEQUENCE {
        IA5String 'rcd'
        SEQUENCE {
          UTF8String '"nam": "James Bond"'
          }
        IA5String 'crn'
        SEQUENCE {
          UTF8String '"For your ears only"'
          }
        }
      }
    }
  mustExclude [2] {
    SEQUENCE {
      IA5String 'attest'
      IA5String 'origid'
      IA5String 'div'
      IA5String 'rph'
      IA5String 'sph'
      IA5String 'rcdi'
      }
    }
  }
```

5.5.1.3. Extended Claims Authorized (Per-TN Subset Constraints)

In the third case, the user is permitted to assert different sets of claims for distinct subsets of the user's authorized TNs. To manage this, the JWTClaimConstraints Authority Token includes permitted values for the claims that are explicitly associated with a specific set of TN(s). This association is achieved by including an EnhancedJWTClaimConstraints permitted value(s) entry for the "orig" claim which identifies the relevant TN(s) to which the other claim values apply.

```
SEQUENCE {
  permittedValues [1] {
    SEQUENCE {
      SEQUENCE {
        IA5String 'rcd'
        SEQUENCE {
          UTF8String '"nam": "James Bond"'
          }
        IA5String 'crn'
        SEQUENCE {
          UTF8String '"For your ears only"'
          }
        IA5String 'orig'
        SEQUENCE {
          UTF8String '"12025551000"'
          UTF8String '"12025551001"'
          }
        }
      }
    }
  mustExclude [2] {
    SEQUENCE {
      IA5String 'attest'
      IA5String 'origid'
      IA5String 'div'
      IA5String 'rph'
      IA5String 'sph'
      IA5String 'rcdi'
      }
    }
  }
```

5.5.2. ACME Procedures when Challenge requires two Authority Tokens

Sections 3 and 4 describe the ACME procedures for issuing a certificate based on a single JWTClaimConstraints identifier. This section describes how these procedures are modified to support the case where the new-order request contains both a TNAuthList and JWTClaimConstraints identifier.

First, the "identifiers" field in the new-order request includes both identifier types:

```
 "identifiers": [
   {"type":"TNAuthList",
    "value":"KHn6xf...jw4A1vgh"}
   {"type":"JWTClaimConstraints",
    "value":"F83n2a...avn27DN3"}]
```

The CA includes two "authorizations" URLs in the 201 (Created) response to the new-order request, one for each identifier:

```
"authorizations": [
   "https://example.com/acme/authz/1234",
   "https://example.com/acme/authz/5678"]
```

The ACME client then queries each "authorizations" URL as shown in Section 4. The CA returns the Authority Token challenge for each identifier. The ACME client responds to each challenge by providing an Authority Token of the appropriate type.

6. Validating the JWTClaimConstraints Authority Token

Upon receiving a response to the challenge, the ACME server MUST perform the following steps to determine the validity of the response:

1. Verify that the value of the "atc" claim is a well-formed JSON object containing the mandatory key values ("tktype", "tkvalue", "fingerprint").

2. Verify Token Issuer: If there is an "x5u" parameter, verify the "x5u" parameter is an HTTPS URL with a reference to a certificate representing the trusted issuer of authority tokens. If there is an "x5c" parameter, verify the certificate array contains a certificate representing the trusted issuer of authority tokens.

3. Verify Signature: Verify the JWTClaimConstraints Authority Token signature using the public key of the certificate referenced by the token’s "x5u" or "x5c" parameter.

4. Verify Token Type: Verify that the "atc" claim contains a "tktype" identifier with the value "JWTClaimConstraints".

5. Verify Constraints Match: Verify that the "atc" claim "tkvalue" identifier contains the equivalent base64url encoded JWTClaimConstraints or EnhancedJWTClaimConstraints certificate extension string value as the Identifier specified in the original challenge.

6. Verify Claims: Verify that the remaining claims are valid (e.g., verify that the token has not expired using the "exp" claim). 7.Verify Account Control: Verify that the "atc" claim "fingerprint" is valid and matches the account key of the client making the request.

7. Verify CA Flag: Verify that the "atc" claim "ca" identifier boolean corresponds to the CA boolean in the Basic Constraints extension in the Certificate Signing Request (CSR) for either a CA certificate or an end-entity certificate.

If all steps in the token validation process pass, then the ACME server MUST set the challenge object "status" to "valid". If any step of the validation process fails, the "status" in the challenge object MUST be set to "invalid".

7. Using ACME-issued Certificates with JSON Web Signature

JSON Web Signature (JWS) objects can include an "x5u" header parameter to refer to a certificate for signature validation. In order to support this usage, the Certificate Authority (CA) MAY host the newly issued certificate and provide a URL that the ACME client owner can directly reference in the "x5u" header parameter of their signed JWS objects.

To facilitate this, the CA MAY add a newly defined field called "x5u" to the 200 (OK) order object response when the certificate is ready for the finalize request:

x5u (optional, string): A URL that can be used to reference the certificate in the "x5u" parameter of a JWS object.

An example of a 200 (OK) response containing the new "x5u" field:

```
HTTP/1.1 200 OK
Content-Type: application/json
Replay-Nonce: CGf81JWBsq8QyIgPCi9Q9X
Link: <https://example.com/acme/directory>;rel="index"
Location: https://example.com/acme/order/TOlocE8rfgo

{
  "status": "valid",
  "expires": "2016-01-20T14:09:07.99Z",

  "notBefore": "2016-01-01T00:00:00Z",
  "notAfter": "2016-01-08T00:00:00Z",

  "identifiers": [
    "type":"JWTClaimConstraints",
    "value":"F83n2a...avn27DN3"
  ],

  "authorizations": ["https://example.com/acme/authz/1234"],

  "finalize": "https://example.com/acme/order/TOlocE8rfgo/finalize",

  "certificate": "https://example.com/acme/cert/mAt3xBGaobw",

  "x5u": "https://example.com/cert-repo/giJI53km23.pem"
}
```

8. Security Considerations

The token represented by this document has the credentials to represent JWTClaimConstraints and EnhancedJWTClaimConstraints, which constrain the resources and claims a certificate holder can assert. The creation, transport, and any storage of this token MUST follow the strictest of security best practices, beyond the recommendations of the use of encrypted transport protocols in this document, to protect it from getting in the hands of bad actors with illegitimate intent to impersonate or misuse the constrained resources.

This document inherits the security properties of [RFC9447]. Implementations SHOULD follow the best practices identified in [RFC8725] for cryptographic security.

This document only specifies SHA256 for the fingerprint hash. However, the syntax of the fingerprint object would permit other algorithms if, due to concerns about algorithmic agility, a more robust algorithm were required at a future time. Future specifications CAN define new algorithms for the fingerprint object as needed.

9. IANA Considerations

This document requests the addition of a new identifier object type to the "ACME Identifier Types" registry defined in Section 9.7.7 of [RFC8555].

Label	Reference
JWTClaimConstraints	RFCThis

10. Acknowledgements

We would like to thank ACME and STIR working groups for valuable contributions to the authority token framework used in this document.

11. Normative 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/rfc/rfc2119>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/rfc/rfc7231>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

[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/rfc/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/rfc/rfc8174>.

[RFC8226] Peterson, J. and S. Turner, "Secure Telephone Identity Credentials: Certificates", RFC 8226, DOI 10.17487/RFC8226, February 2018, <https://www.rfc-editor.org/rfc/rfc8226>.

[RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <https://www.rfc-editor.org/rfc/rfc8555>.

[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/rfc/rfc8725>.

[RFC9060] Peterson, J., "Secure Telephone Identity Revisited (STIR) Certificate Delegation", RFC 9060, DOI 10.17487/RFC9060, September 2021, <https://www.rfc-editor.org/rfc/rfc9060>.

[RFC9118] Housley, R., "Enhanced JSON Web Token (JWT) Claim Constraints for Secure Telephone Identity Revisited (STIR) Certificates", RFC 9118, DOI 10.17487/RFC9118, August 2021, <https://www.rfc-editor.org/rfc/rfc9118>.

[RFC9447] Peterson, J., Barnes, M., Hancock, D., and C. Wendt, "Automated Certificate Management Environment (ACME) Challenges Using an Authority Token", RFC 9447, DOI 10.17487/RFC9447, September 2023, <https://www.rfc-editor.org/rfc/rfc9447>.

[RFC9448] Wendt, C., Hancock, D., Barnes, M., and J. Peterson, "TNAuthList Profile of Automated Certificate Management Environment (ACME) Authority Token", RFC 9448, DOI 10.17487/RFC9448, September 2023, <https://www.rfc-editor.org/rfc/rfc9448>.

Authors' Addresses

Chris Wendt
Somos Inc.
United States of America
Email: chris@appliedbits.com

David Hancock
Somos Inc.
United States of America
Email: davidhancock.ietf@gmail.com