Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 28, 2017

J. Bradley
Ping Identity
P. Hunt
Oracle Corporation
M. Jones
Microsoft
H. Tschofenig
ARM Limited

February 24, 2017

OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution
draft-ietf-oauth-pop-key-distribution-03

Abstract

RFC 6750 specified the bearer token concept for securing access to protected resources. Bearer tokens need to be protected in transit as well as at rest. When a client requests access to a protected resource it hands-over the bearer token to the resource server.

The OAuth 2.0 Proof-of-Possession security concept extends bearer token security and requires the client to demonstrate possession of a key when accessing a protected resource.

This document describes how the client obtains this keying material from the authorization server.

Status of This Memo

Table of Contents

1. Introduction

The work on additional security mechanisms beyond OAuth 2.0 bearer tokens [12] is motivated in [17], which also outlines use cases, requirements and an architecture. This document defines the ability for the client indicate support for this functionality and to obtain keying material from the authorization server. As an outcome of the

exchange between the client and the authorization server is an access token that is bound to keying material. Clients that access protected resources then need to demonstrate knowledge of the secret key that is bound to the access token.

To best describe the scope of this specification, the OAuth 2.0 protocol exchange sequence is shown in Figure 1. The extension defined in this document piggybacks on the message exchange marked with (C) and (D).

```
     +--------+                               +---------------+
     |        |--(A)- Authorization Request ->|   Resource    |
     |        |                               |     Owner     |
     |        |<-(B)-- Authorization Grant ---|               |
     |        |                               +---------------+
     |        |
     |        |                               +---------------+
     |        |--(C)-- Authorization Grant -->| Authorization |
     | Client |                               |     Server    |
     |        |<-(D)----- Access Token -------|               |
     |        |                               +---------------+
     |        |
     |        |                               +---------------+
     |        |--(E)----- Access Token ------>|    Resource   |
     |        |                               |     Server    |
     |        |<-(F)--- Protected Resource ---|               |
     +--------+                               +---------------+
```

Figure 1: Abstract OAuth 2.0 Protocol Flow

In OAuth 2.0 [2] access tokens can be obtained via authorization grants and using refresh tokens. The core OAuth specification defines four authorization grants, see Section 1.3 of [2], and [14] adds an assertion-based authorization grant to that list. The token endpoint, which is described in Section 3.2 of [2], is used with every authorization grant except for the implicit grant type. In the implicit grant type the access token is issued directly.

This document extends the functionality of the token endpoint, i.e., the protocol exchange between the client and the authorization server, to allow keying material to be bound to an access token. Two types of keying material can be bound to an access token, namely symmetric keys and asymmetric keys. Conveying symmetric keys from the authorization server to the client is described in Section 4 and the procedure for dealing with asymmetric keys is described in Section 5.

2. Terminology

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this specification are to be interpreted as described in [1].

Session Key:

The term session key refers to fresh and unique keying material established between the client and the resource server. This session key has a lifetime that corresponds to the lifetime of the access token, is generated by the authorization server and bound to the access token.

This document uses the following abbreviations:

JWA:	JSON Web Algorithms (JWA) [7]
JWT:	JSON Web Token (JWT) [9]
JWS:	JSON Web Signature (JWS) [6]
JWK:	JSON Web Key (JWK) [5]
JWE:	JSON Web Encryption (JWE) [8]

3. Audience

When an authorization server creates an access token, according to the PoP security architecture [17], it may need to know which resource server will process it. This information is necessary when the authorization server applies integrity protection to the JWT using a symmetric key and has to selected the key of the resource server that has to verify it. The authorization server also requires this audience information if it has to encrypt a symmetric session key inside the access token using a long-term symmetric key.

This section defines a new header that is used by the client to indicate what protected resource at which resource server it wants to access. This information may subsequently also communicated by the authorization server securely to the resource server, for example within the audience field of the access token.

QUESTION: A benefit of asymmetric cryptography is to allow clients to request a PoP token for use with multiple resource servers. The downside of that approach is linkability since different resource servers will be able to link individual requests to the same client.

(The same is true if the a single public key is linked with PoP tokens used with different resource servers.) Nevertheless, to support the functionality the audience parameter could carry an array of values. Is this desirable?

3.1. Audience Parameter

The client constructs the access token request to the token endpoint by adding the 'aud' parameter using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body.

The URI included in the aud parameter MUST be an absolute URI as defined by Section 4.3 of [3]. It MAY include an "application/x-www-form-urlencoded" formatted query component (Section 3.4 of [3]). The URI MUST NOT include a fragment component.

The ABNF syntax for the 'aud' element is defined in Appendix A.

3.2. Processing Instructions

Step (0): As an initial step the client typically determines the resource server it wants to interact with. This may, for example, happen as part of a discovery procedure or via manual configuration.

Step (1): The client starts the OAuth 2.0 protocol interaction based on the selected grant type.

Step (2): When the client interacts with the token endpoint to obtain an access token it MUST populate the newly defined 'audience' parameter with the information obtained in step (0).

Step (2): The authorization server who obtains the request from the client needs to parse it to determine whether the provided audience value matches any of the resource servers it has a relationship with. If the authorization server fails to parse the provided value it MUST reject the request using an error response with the error code "invalid_request". If the authorization server does not consider the resource server acceptable it MUST return an error response with the error code "access_denied". In both cases additional error information may be provided via the error_description, and the error_uri parameters. If the request has, however, been verified successfully then the authorization server MUST include the audience claim into the access token with the value copied from the audience field provided by the client. In case the access token is encoded using the JSON Web Token format [9] the "aud" claim MUST be used. The access token, if

passed per value, MUST be protected against modification by either using a digital signature or a keyed message digest. Access tokens can also be passed by reference, which then requires the token introspection endpoint (or a similiar, proprietary protocol mechanism) to be used. The authorization server returns the access token to the client, as specified in [2].

Subsequent steps for the interaction between the client and the resource server are beyond the scope of this document.

4. Symmetric Key Transport

4.1. Client-to-AS Request

In case a symmetric key shall be bound to an PoP token the following procedure is applicable. In the request message from the OAuth client to the OAuth authorization server the following parameters MAY be included:

token_type: OPTIONAL. See Section 6 for more details.

alg: OPTIONAL. See Section 6 for more details.

These two new parameters are optional in the case where the authorization server has prior knowledge of the capabilities of the client otherwise these two parameters are required. This prior knowledge may, for example, be set by the use of a dynamic client registration protocol exchange.

QUESTION: Should we register these two parameters for use with the dynamic client registration protocol?

For example, the client makes the following HTTP request using TLS (extra line breaks are for display purposes only).

```
        POST /token HTTP/1.1
        Host: server.example.com
        Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
        Content-Type: application/x-www-form-urlencoded;charset=UTF-8

        grant_type=authorization_code
        &code=SplxlOBeZQQYbYS6WxSbIA
        &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
        &token_type=pop
        &alg=HS256
```

Example Request to the Authorization Server

4.2. Client-to-AS Response

If the access token request has been successfully verified by the authorization server and the client is authorized to obtain a PoP token for the indicated resource server, the authorization server issues an access token and optionally a refresh token. If client authentication failed or is invalid, the authorization server returns an error response as described in Section 5.2 of [2].

The authorization server MUST include an access token and a 'key' element in a successful response. The 'key' parameter either contains a plain JWK structure or a JWK encrypted with a JWE. The difference between the two approaches is the following:

Plain JWK: If the JWK container is placed in the 'key' element then the security of the overall PoP architecture relies on Transport Layer Security (TLS) between the authorization server and the client. Figure 2 illustrates an example response using a plain JWK for key transport from the authorization server to the client.

JWK protected by a JWE: If the JWK container is protected by a JWE then additional security protection at the application layer is provided between the authorization server and the client beyond the use of TLS. This approach is a reasonable choice, for example, when a hardware security module is available on the client device and confidentiality protection can be offered directly to this hardware security module.

Note that there are potentially two JSON-encoded structures in the response, namely the access token (with the recommended JWT encoding) and the actual key transport mechanism itself. Note, however, that the two structures serve a different purpose and are consumed by different parites. The access token is created by the authorization server and processed by the resource server (and opaque to the

client) whereas the key transport payload is created by the authorization server and processed by the client; it is never forwarded to the resource server.

```
  HTTP/1.1 200 OK
  Content-Type: application/json
  Cache-Control: no-store

  {
    "access_token":"SlAV32hkKG ...
     (remainder of JWT omitted for brevity;
     JWT contains JWK in the cnf claim)",
    "token_type":"pop",
    "expires_in":3600,
    "refresh_token":"8xLOxBtZp8",
    "key":"eyJhbGciOiJSU0ExXzUi ...
     (remainder of plain JWK omitted for brevity)"
  }
```

Figure 2: Example: Response from the Authorization Server (Symmetric Variant)

The content of the key parameter, which is a JWK in our example, is shown in Figure 3.

```
  {
   "kty":"oct",
   "kid":"id123",
   "alg":"HS256",
   "k":"ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE"
  }
```

Figure 3: Example: Key Transport to Client via a JWK

The content of the 'access_token' in JWT format contains the 'cnf' (confirmation) claim, as shown in Figure 4. The confirmation claim is defined in [10]. The digital signature or the keyed message digest offering integrity protection is not shown in this example but MUST be present in a real deployment to mitigate a number of security threats. Those security threats are described in [17].

The JWK in the key element of the response from the authorization server, as shown in Figure 2, contains the same session key as the JWK inside the access token, as shown in Figure 4. It is, in this

example, protected by TLS and transmitted from the authorization server to the client (for processing by the client).

```
   {
      "iss": "https://server.example.com",
      "sub": "24400320",
      "aud": "s6BhdRkqt3",
      "nonce": "n-0S6_WzA2Mj",
      "exp": 1311281970,
      "iat": 1311280970,
      "cnf":{
        "jwk":
          "JDLUhTMjU2IiwiY3R5Ijoi ...
           (remainder of JWK protected by JWE omitted for brevity)"
        }
   }
```

Figure 4: Example: Access Token in JWT Format

Note: When the JWK inside the access token contains a symmetric key it MUST be confidentiality protected using a JWE to maintain the security goals of the PoP architecture, as described in [17] since content is meant for consumption by the selected resource server only.

Note: This document does not impose requirements on the encoding of the access token. The examples used in this document make use of the JWT structure since this is the only standardized format.

If the access token is only a reference then a look-up by the resource server is needed, as described in the token introspection specification [18].

5. Asymmetric Key Transport

5.1. Client-to-AS Request

In case an asymmetric key shall be bound to an access token then the following procedure is applicable. In the request message from the OAuth client to the OAuth authorization server the request MAY include the following parameters:

token_type: OPTIONAL. See Section 6 for more details.

alg: OPTIONAL. See Section 6 for more details.

key: OPTIONAL. This field contains information about the public key the client would like to bind to the access token in the JWK format. If the client does not provide a public key then the authorization server MUST create an ephemeral key pair (considering the information provided by the client) or alternatively respond with an error message. The client may also convey the fingerprint of the public key to the authorization server instead of passing the entire public key along (to conserve bandwidth). [11] defines a way to compute a thumbprint for a JWK and to embedd it within the JWK format.

The 'token_type' and the 'alg' parameters are optional in the case where the authorization server has prior knowledge of the capabilities of the client otherwise these two parameters are required.

For example, the client makes the following HTTP request using TLS (extra line breaks are for display purposes only) shown in Figure 5.

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&token_type=pop
&alg=RS256
&key=eyJhbGciOiJSU0ExXzUi ...
(remainder of JWK omitted for brevity)
```

Figure 5: Example Request to the Authorization Server (Asymmetric Key Variant)

As shown in Figure 6 the content of the 'key' parameter contains the RSA public key the client would like to associate with the access token.

```
      {"kty":"RSA",
       "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx
        4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMs
        tn64tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2
        QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbI
        SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqb
        w0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
        "e":"AQAB",
        "alg":"RS256",
        "kid":"id123"}
```

Figure 6: Client Providing Public Key to Authorization Server

5.2. Client-to-AS Response

If the access token request is valid and authorized, the authorization server issues an access token and optionally a refresh token. If the request client authentication failed or is invalid, the authorization server returns an error response as described in Section 5.2 of [2].

The authorization server also places information about the public key used by the client into the access token to create the binding between the two. The new token type "public_key" is placed into the 'token_type' parameter.

An example of a successful response is shown in Figure 7.

```
        HTTP/1.1 200 OK
        Content-Type: application/json;charset=UTF-8
        Cache-Control: no-store
        Pragma: no-cache

        {
          "access_token":"2YotnFZFE....jr1zCsicMWpAA",
          "token_type":"pop",
          "alg":"RS256",
          "expires_in":3600,
          "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
        }
```

Figure 7: Example: Response from the Authorization Server (Asymmetric Variant)

The content of the 'access_token' field contains an encoded JWT with the following structure, as shown in Figure 8. The digital signature

or the keyed message digest offering integrity protection is not shown (but must be present).

```
    {
      "iss":"xas.example.com",
      "aud":"http://auth.example.com",
      "exp":"1361398824",
      "nbf":"1360189224",
      "cnf":{
        "jwk":{"kty":"RSA",
          "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx
     4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMs
     tn64tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2
     QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbI
     SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqb
     w0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
          "e":"AQAB",
          "alg":"RS256",
          "kid":"id123"}
       }
     }
```

Figure 8: Example: Access Token Structure (Asymmetric Variant)

Note: In this example there is no need for the authorization server to convey further keying material to the client since the client is already in possession of the private RSA key.

6. Token Types and Algorithms

To allow clients to indicate support for specific token types and respective algorithms they need to interact with authorization servers. They can either provide this information out-of-band, for example, via pre-configuration or up-front via the dynamic client registration protocol [16].

The value in the 'alg' parameter together with value from the 'token_type' parameter allow the client to indicate the supported algorithms for a given token type. The token type refers to the specification used by the client to interact with the resource server to demonstrate possession of the key. The 'alg' parameter provides further information about the algorithm, such as whether a symmetric or an asymmetric crypto-system is used. Hence, a client supporting a specific token type also knows how to populate the values to the 'alg' parameter.

The value for the 'token_type' MUST be taken from the 'OAuth Access Token Types' registry created by [2].

This document does not register a new value for the OAuth Access Token Types registry nor does it define values to be used for the 'alg' parameter since this is the responsibility of specifications defining the mechanism for clients interacting with resource servers. An example of such specification can be found in [19].

The values in the 'alg' parameter are case-sensitive. If the client supports more than one algorithm then each individual value MUST be separated by a space.

7. Security Considerations

[17] describes the architecture for the OAuth 2.0 proof-of-possession security architecture, including use cases, threats, and requirements. This requirements describes one solution component of that architecture, namely the mechanism for the client to interact with the authorization server to either obtain a symmetric key from the authorization server, to obtain an asymmetric key pair, or to offer a public key to the authorization. In any case, these keys are then bound to the access token by the authorization server.

To summarize the main security recommendations: A large range of threats can be mitigated by protecting the contents of the access token by using a digital signature or a keyed message digest. Consequently, the token integrity protection MUST be applied to prevent the token from being modified, particularly since it contains a reference to the symmetric key or the asymmetric key. If the access token contains the symmetric key (see Section 2.2 of [10] for a description about how symmetric keys can be securely conveyed within the access token) this symmetric key MUST be encrypted by the authorization server with a long-term key shared with the resource server.

To deal with token redirect, it is important for the authorization server to include the identity of the intended recipient (the audience), typically a single resource server (or a list of resource servers), in the token. Using a single shared secret with multiple authorization server to simplify key management is NOT RECOMMENDED since the benefit from using the proof-of-possession concept is significantly reduced.

Token replay is also not possible since an eavesdropper will also have to obtain the corresponding private key or shared secret that is bound to the access token. Nevertheless, it is good practice to

limit the lifetime of the access token and therefore the lifetime of associated key.

The authorization server MUST offer confidentiality protection for any interactions with the client. This step is extremely important since the client will obtain the session key from the authorization server for use with a specific access token. Not using confidentiality protection exposes this secret (and the access token) to an eavesdropper thereby making the OAuth 2.0 proof-of-possession security model completely insecure. OAuth 2.0 [2] relies on TLS to offer confidentiality protection and additional protection can be applied using the JWK [5] offered security mechanism, which would add an additional layer of protection on top of TLS for cases where the keying material is conveyed, for example, to a hardware security module. Which version(s) of TLS ought to be implemented will vary over time, and depend on the widespread deployment and known security vulnerabilities at the time of implementation. At the time of this writing, TLS version 1.2 [4] is the most recent version. The client MUST validate the TLS certificate chain when making requests to protected resources, including checking the validity of the certificate.

Similarly to the security recommendations for the bearer token specification [12] developers MUST ensure that the ephemeral credentials (i.e., the private key or the session key) is not leaked to third parties. An adversary in possession of the ephemeral credentials bound to the access token will be able to impersonate the client. Be aware that this is a real risk with many smart phone app and Web development environments.

Clients can at any time request a new proof-of-possession capable access token. Using a refresh token to regularly request new access tokens that are bound to fresh and unique keys is important. Keeping the lifetime of the access token short allows the authorization server to use shorter key sizes, which translate to a performance benefit for the client and for the resource server. Shorter keys also lead to shorter messages (particularly with asymmetric keying material).

When authorization servers bind symmetric keys to access tokens then they SHOULD scope these access tokens to a specific permissions.

8. IANA Considerations

This specification registers the following parameters in the OAuth Parameters Registry established by [2].

Parameter name: alg

Parameter usage location: token request, token response,
   authorization response

Change controller: IETF

Specification document(s): [[this document]]

Related information: None

Parameter name: key

Parameter usage location: token request, token response,
   authorization response

Change controller: IETF

Specification document(s): [[this document]]

Related information: None

Parameter name: aud

Parameter usage location: token request

Change controller: IETF

Specification document(s): [[This document.]

Related information: None

9. Acknowledgements

We would like to thank Chuck Mortimore for his review comments.

10. References

10.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[2] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[3] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[4] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[5] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <http://www.rfc-editor.org/info/rfc7517>.

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

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

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

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

[10] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <http://www.rfc-editor.org/info/rfc7800>.

[11] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <http://www.rfc-editor.org/info/rfc7638>.

10.2. Informative References

[12] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <http://www.rfc-editor.org/info/rfc6750>.

[13] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[14] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <http://www.rfc-editor.org/info/rfc7521>.

[15] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <http://www.rfc-editor.org/info/rfc7636>.

[16] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <http://www.rfc-editor.org/info/rfc7591>.

[17] Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-08 (work in progress), July 2016.

[18] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <http://www.rfc-editor.org/info/rfc7662>.

[19] Richer, J., Bradley, J., and H. Tschofenig, "A Method for Signing HTTP Requests for OAuth", draft-ietf-oauth-signed-http-request-03 (work in progress), August 2016.

Appendix A. Augmented Backus-Naur Form (ABNF) Syntax

This section provides Augmented Backus-Naur Form (ABNF) syntax descriptions for the elements defined in this specification using the notation of [13].

A.1. 'aud' Syntax

The ABNF syntax is defined as follows where by the "URI-reference" definition is taken from [3]:

```
aud = URI-reference
```

A.2. 'key' Syntax

The "key" element is defined in Section 4 and Section 5:

```
key = 1*VSCHAR
```

A.3. 'alg' Syntax

The "alg" element is defined in Section 6:

```
alg = alg-token *( SP alg-token )

alg-token = 1*NQCHAR
```

Authors' Addresses

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Phil Hunt
Oracle Corporation

Email: phil.hunt@yahoo.com
URI: http://www.indepdentid.com

Michael B. Jones
Microsoft

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

Hannes Tschofenig
ARM Limited
Austria

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at