Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 24, 2014

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

April 22, 2014

OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution

draft-bradley-oauth-pop-key-distribution-00.txt

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 since the security model is based on proof-of-possession.

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

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 http://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 October 24, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

The work on additional security mechanisms beyond OAuth 2.0 bearer tokens [11] 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).

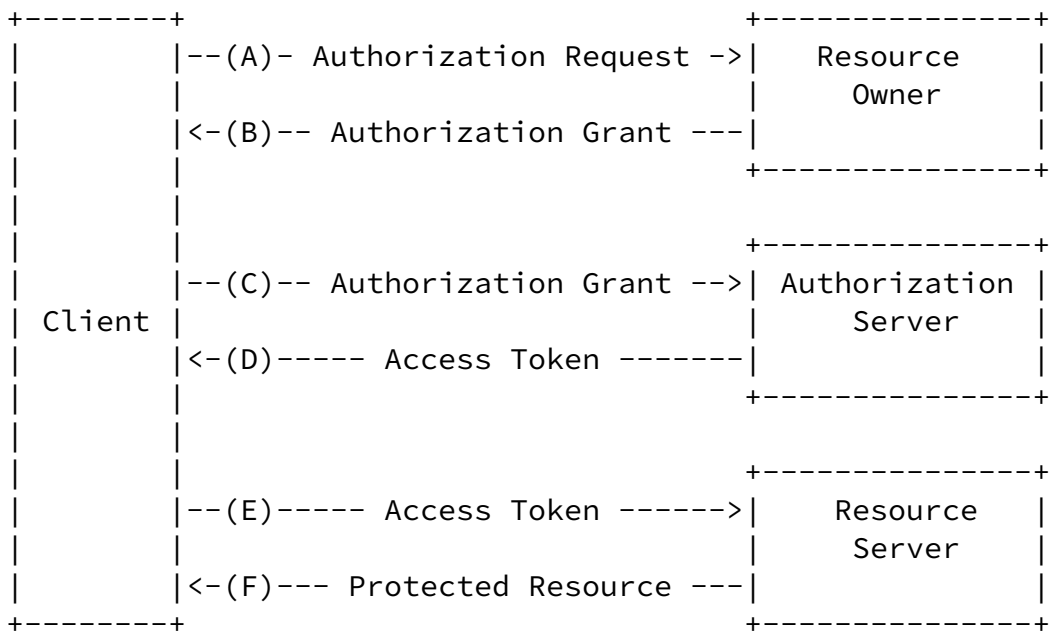

Figure 1: Abstract OAuth 2.0 Protocol Flow

OAuth 2.0 [2] offers different ways to obtain an access token, namely using authorization grants and using a refresh token. 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.

QUESTION: This document focuses on binding the keys to access tokens only. Keys can, however, also be bound to refresh tokens and to the authorization code, as described in [15]. Should the scope of this document be extended?

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 a corresponding key based on the resource server that has to verify the signature. The authorization server also requires this audience information if it embeds a session key inside the access token and if this symmetric key is encrypted with a 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: With the use of public key cryptography the client might request a PoP token for use with multiple resource servers. 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, for example, as part of a discovery procedure.

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 MUST be protected against modification by either using a digital signature or a keyed message digest. 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 access 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.

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 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 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 protection 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 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 trusted security module is available on the client device and the confidentiality protection is offered directly to this trusted element.

Note that there are potentially two JSON-encoded structures in the response, namely the access token that may utilize the JWT format and the actual key transport mechanism itself.


```
  HTTP/1.1 200 OK
```

```
Content-Type: application/json
Cache-Control: no-store

{
  "access_token":"SlAV32hkKG ...
    ... (remainder of JWT omitted for brevity)",
  "token_type":"pop",
  "expires_in":3600,
  "refresh_token":"8xLOxBtZp8",
  "key":"eyJhbGciOiJSU0ExXzUi ...
    ... (remainder of JWK/JWE 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",
 "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' element, as shown in Figure 4. 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.

The JWK in the key element of the response from the authorization server shown in Figure 2 contains the same session key as the JWK inside the access token shown in Figure 4 but 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 JWE omitted for brevity)"
     }
}
```

Figure 4: Example: Access Token in JWT Format

Note: The JWK content inside the access token is meant for consumption by the resource server. It MUST be confidentiality protected using a JWE.

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 access tokens are conveyed per value between the authorization server, the client, and the resource server.

If the access token is only a handle to the access token itself 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 following parameters 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 JSON

Web Key format. If the client does not provide a public key then the authorization server MUST create an ephemeral key pair

(considering the profile information provided by the client) or alternative respond with an error message.

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)

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

```
{"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":"client@example.com"}
```

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":"client@example.com"}
   }
 }
```

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.

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 Access Token Types registry nor does it define values to be used for the alg parameter. Instead this is the responsibility of specifications

defining the mechanism for clients interacting with resource servers, such as [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 JSON Web Key (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 [11] 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 MUST scope these access tokens to a specific resource server.

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

Add your name here.

10. References

10.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[3] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[4] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[5] Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-25 (work in progress), March 2014.

[6] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-25 (work in progress), March 2014.

[7] Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms-25 (work in progress), March 2014.

[8] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-25 (work in progress), March 2014.

[9] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-19 (work in progress), March 2014.

[10] Jones, M., Bradley, J., and H. Tschofenig, "Proof-Of-Possession Semantics for JSON Web Tokens (JWTs)", draft-jones-oauth-proof-of-possession-00 (work in progress), April 2014.

10.2. Informative References

[11] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012.

[12] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[13] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[14] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", draft-ietf-oauth-assertions-15 (work in progress), March 2014.

[15] Sakimura, N., Bradley, J., and N. Agarwal, "OAuth Symmetric Proof of Posession for Code Extension", draft-sakimura-oauth-tcse-03 (work in progress), April 2014.

[16] Richer, J., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Core Protocol", draft-ietf-oauth-dyn-reg-16 (work in progress), February 2014.

[17] Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-hunt-oauth-pop-architecture-00 (work in progress), April 2014.

[18] Richer, J., "OAuth Token Introspection", draft-richer-oauth-introspection-04 (work in progress), May 2013.

[19] Richer, J., "A Method for Signing an HTTP Requests for OAuth", April 2014.

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