Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2019

J. Bradley
Ping Identity
P. Hunt
Oracle Corporation
M. Jones
Microsoft
H. Tschofenig
Arm Ltd.
M. Meszaros
GITDA

March 11, 2019

OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution
draft-ietf-oauth-pop-key-distribution-05

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 requests and obtains a PoP access token from the authorization server for use with HTTPS-based transport. Alternative transports, for example using the Constrained Application Protocol (CoAP), have been specified in companion specifications.

This Internet-Draft will expire on September 12, 2019.

Table of Contents

1. Introduction

The work on proof-of-possession tokens, an extended token security mechanisms for OAuth 2.0, is motivated in [21]. This document defines the ability for the client request and to obtain PoP tokens from the authorization server. After successfully completing the exchange the client is in possession of a PoP token and the keying material bound to it. Clients that access protected resources then need to demonstrate knowledge of the secret key that is bound to the PoP 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). To demonstrate possession of the private/secret key to the resource server protocol mechanisms outside the scope of this document are used.

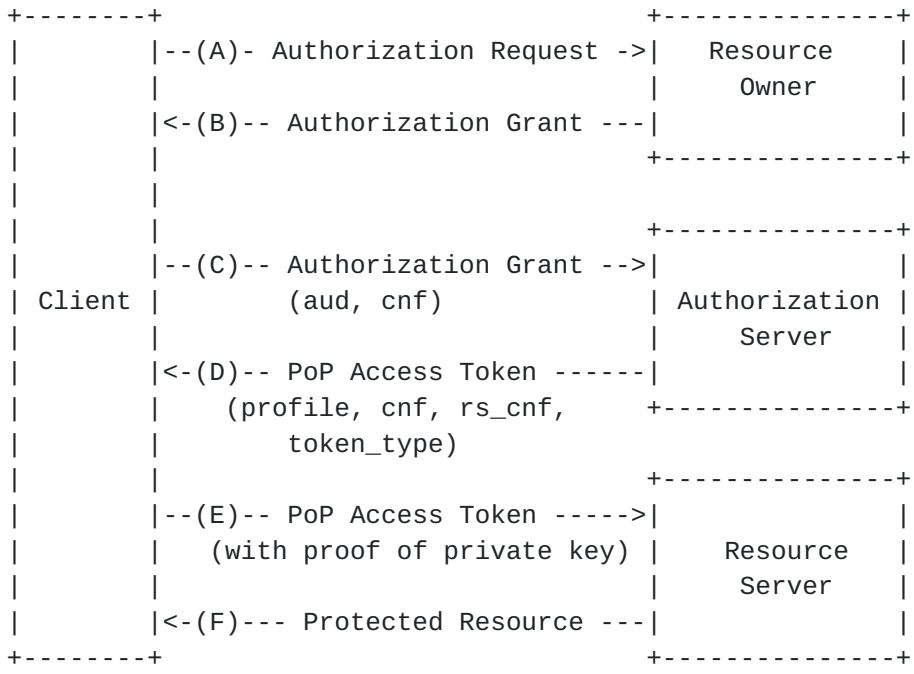

Figure 1: Augmented OAuth 2.0 Protocol Flow

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 [18] 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 4.1 and the procedure for dealing with asymmetric keys is described in Section 4.2.

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:

In the context of this specification '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[7]

JWT: JSON Web Token[9]

JWS: JSON Web Signature[6]

JWK: JSON Web Key[5]

JWE: JSON Web Encryption[8]

CWT: CBOR Web Token[14]

COSE: CBOR Object Signing and Encryption[15]

3. 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 follow the processing steps for the HTTPS-based transport described in Section 5.6.1 of [12]. This includes determining whether to use the 'aud' and the 'cnf' parameters.

Step (3): The authorization server parses the request from the server and determines the suitable response based on the

description in Section 5.6.2 of [12]. In case of an error the procedure in Section 5.6.3 of [12] is applicable.

Note that PoP access tokens may be encoded in a variety of ways. In case the access token is encoded using the JSON Web Token (JWT) format [9] it MUST be protected against modification by either using a digital signature or a keyed message digest, as described in [6]. The JWT may also be encrypted using [8]. [14] defines an alternative token format based on CBOR. The proof-of-possession token functionality is defined in [13]. Finally, access tokens can also be passed by reference, which then requires the token introspection endpoint (or a similiar, proprietary protocol mechanism) to be used. This specification does not mandate a specific PoP token format.

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

4. Examples

This section provides a number of examples.

4.1. Symmetric Key Transport

4.1.1. Client-to-AS Request

The client starts with a request to the authorization server indicating that it is interested to obtain a token for example-server.

```
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
&aud=example-server
```

Example Request to the Authorization Server

4.1.2. 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.

Figure 2 shows a response containing a token and a "cnf" parameter with a symmetric proof-of-possession key.

```
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",
  "cnf":{
     "keys" : {
       "kty" : "Symmetric",
       "kid" : b64'39Gqlw',
       "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
     }
  }
}
```

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

Note that the cnf payload in Figure 2 is not encrypted at the application layer since Transport Layer Security is used between the AS and the client and the content of the cnf payload is consumed by the client itself. Alternatively, a JWE could be used to encrypt the key distribution, as shown in Figure 3.

```
{
  "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",
  "cnf":{
      "jwe":
        "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkExMjhDQkMtSFMyNTYifQ.
        (remainder of JWE omitted for brevity)"
      }
  }
}
```

Figure 3: Example: Encrypted Symmmetric Key

The content of the key parameter, which is a JWK in our example, is shown in Figure 4.

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

Figure 4: 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 5. 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 has to 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, as shown in Figure 2, contains the same session key as the JWK inside the access token, as shown in Figure 5. 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 5: 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 [21] 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 but the CWT format is an equally appropriate format to use.

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 [22].

4.2. Asymmetric Key Transport

4.2.1. Client-to-AS Request

This example illustrates the case where an asymmetric key shall be bound to an access token. The client makes the following HTTPS request shown in Figure 6. 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
&cnf=eyJhbGciOiJSU0ExXzUi ...
(remainder of JWK omitted for brevity)
```

Figure 6: Example Request to the Authorization Server (Asymmetric Key Variant)

As shown in Figure 7 the content of the 'cnf' parameter contains the ECC public key the client would like to associate with the access token.

```
"jwk":{
  "kty": "EC",
  "use": "sig",
  "crv": "P-256",
  "x": "18wHLeIgW9wVN6VD1Txgpqy2LszYkMf6J8njVAibvhM",
  "y": "-V4dS4UaLMgP_4fY4j8ir7cl1TXlFdAgcx55o7TkcSA"
 }
```

Figure 7: Client Providing Public Key to Authorization Server

4.2.2. 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. 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 8.

```
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",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

Figure 8: Example: Response from the Authorization Server (Asymmetric Variant)

The content of the 'access_token' field contains an encoded JWT, as shown in Figure 9. The digital signature covering the access token offering authenticity and integrity protection is not shown below (but must be present).

```
{
  "iss":"xas.example.com",
  "aud":"http://auth.example.com",
  "exp":"1361398824",
  "nbf":"1360189224",
  "cnf":{
     "jwk" : {
       "kty" : "EC",
       "kid" : h'11',
       "crv" : "P-256",
       "x" : b64'usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8',
       "y" : b64'IBOL+C3BttVivg+lSreASjpkttcsz+1rb7btKLv8EX4'
     }
   }
}
```

Figure 9: 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 key.

5. Security Considerations

[21] 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 [16] 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.

6. IANA Considerations

This document does not require any actions by IANA.

7. Acknowledgements

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

8. References

8.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[2] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc5246>.

[5] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://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, <https://www.rfc-editor.org/info/rfc7515>.

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

[8] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7638>.

[12] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-22 (work in progress), March 2019.

[13] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-06 (work in progress), February 2019.

[14] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[15] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

8.2. Informative References

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

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

[18] 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, <https://www.rfc-editor.org/info/rfc7521>.

[19] 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, <https://www.rfc-editor.org/info/rfc7636>.

[20] 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, <https://www.rfc-editor.org/info/rfc7591>.

[21] 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.

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

[23] IANA, "JSON Web Token Claims", March 2019.

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 Ltd.
Absam 6067
Austria

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

Mihaly Meszaros
GITDA
Debrecen 4033
Hungary

Email: bakfitty@gmail.com
URI: https://github.com/misi