Workgroup: Web Authorization Protocol
Internet-Draft: draft-ietf-oauth-dpop-02
Published: 18 November 2020
Intended Status: Standards Track
Expires: 22 May 2021
Authors: D. Fett B. Campbell J. Bradley
yes.com Ping Identity Yubico
T. Lodderstedt M. Jones D. Waite
yes.com Microsoft Ping Identity

OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer (DPoP)

Abstract

This document describes a mechanism for sender-constraining OAuth 2.0 tokens via a proof-of-possession mechanism on the application level. This mechanism allows for the detection of replay attacks with access and refresh tokens.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

DPoP, an abbreviation for Demonstrating Proof-of-Possession at the Application Layer, is an application-level mechanism for sender-constraining OAuth access and refresh tokens. It enables a client to demonstrate proof-of-possession of a public/private key pair by including a DPoP header in an HTTP request. The value of the header is a JWT [RFC7519] that enables the authorization server to bind issued tokens to the public part of the client's key pair. Recipients of such tokens are then able to verify the binding of the token to the key pair that the client has demonstrated that it holds via the DPoP header, thereby providing some assurance that the client presenting the token also possesses the private key. In other words, the legitimate presenter of the token is constrained to be the sender that holds and can prove possession of the private part of the key pair.

The mechanism described herein can be used in cases where other methods of sender-constraining tokens that utilize elements of the underlying secure transport layer, such as [RFC8705] or [I-D.ietf-oauth-token-binding], are not available or desirable. For example, due to a sub-par user experience of TLS client authentication in user agents and a lack of support for HTTP token binding, neither mechanism can be used if an OAuth client is a Single Page Application (SPA) running in a web browser. Native applications installed and run on a user's device, which often have dedicated protected storage for cryptographic keys. are another example well positioned to benefit from DPoP-bound tokens to guard against misuse of tokens by a compromised or malicious resource.

DPoP can be used to sender-constrain access tokens regardless of the client authentication method employed. Furthermore, DPoP can also be used to sender-constrain refresh tokens issued to public clients (those without authentication credentials associated with the client_id).

1.1. Conventions and 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.

This specification uses the terms "access token", "refresh token", "authorization server", "resource server", "authorization endpoint", "authorization request", "authorization response", "token endpoint", "grant type", "access token request", "access token response", and "client" defined by The OAuth 2.0 Authorization Framework [RFC6749].

2. Objectives

The primary aim of DPoP is to prevent unauthorized or illegitimate parties from using leaked or stolen access tokens by binding a token to a public key upon issuance and requiring that the client demonstrate possession of the corresponding private key when using the token. This constrains the legitimate sender of the token to only the party with access to the private key and gives the server receiving the token added assurances that the sender is legitimately authorized to use it.

Access tokens that are sender-constrained via DPoP thus stand in contrast to the typical bearer token, which can be used by any party in possession of such a token. Although protections generally exist to prevent unintended disclosure of bearer tokens, unforeseen vectors for leakage have occurred due to vulnerabilities and implementation issues in other layers in the protocol or software stack (CRIME, BREACH, Heartbleed, and the Cloudflare parser bug are some examples). There have also been numerous published token theft attacks on OAuth implementations themselves. DPoP provides a general defense in depth against the impact of unanticipated token leakage. DPoP is not, however, a substitute for a secure transport and MUST always be used in conjunction with HTTPS.

The very nature of the typical OAuth protocol interaction necessitates that the client disclose the access token to the protected resources that it accesses. The attacker model in [I-D.ietf-oauth-security-topics] describes cases where a protected resource might be counterfeit, malicious or compromised and play received tokens against other protected resources to gain unauthorized access. Properly audience restricting access tokens can prevent such misuse, however, doing so in practice has proven to be prohibitively cumbersome (even despite extensions such as [RFC8707]) for many deployments. Sender-constraining access tokens is a more robust and straightforward mechanism to prevent such token replay at a different endpoint and DPoP is an accessible application layer means of doing so.

Due to the potential for cross-site scripting (XSS), browser-based OAuth clients bring to bear added considerations with respect to protecting tokens. The most straightforward XSS-based attack is for an attacker to exfiltrate a token and use it themselves completely independent from the legitimate client. A stolen access token is used for protected resource access and a stolen refresh token for obtaining new access tokens. If the private key is non-extractable (as is possible with [W3C.WebCryptoAPI]), DPoP renders exfiltrated tokens alone unusable.

XXS vulnerabilities also allow an attacker to execute code in the context of the browser-based client application and maliciously use a token indirectly through the the client. That execution context has access to utilize the signing key and thus can produce DPoP proofs to use in conjunction with the token. At this application layer there is most likely no feasible defense against this threat except generally preventing XSS, therefore it is considered out of scope for DPoP.

Malicious XSS code executed in the context of the browser-based client application is also in a position to create DPoP proofs with timestamp values in the future and exfiltrate them in conjunction with a token. These stolen artifacts can later be used together independent of the client application to access protected resources. The impact of such precomputed DPoP proofs can be limited somewhat by a browser-based client generating and using a new DPoP key for each new authorization code grant.

Additional security considerations are discussed in Section 8.

3. Concept

The main data structure introduced by this specification is a DPoP proof JWT, described in detail below, which is sent as a header in an HTTP request. A client uses a DPoP proof JWT to prove the possession of a private key corresponding to a certain public key. Roughly speaking, a DPoP proof is a signature over a timestamp and some data of the HTTP request to which it is attached.

```
+--------+                                          +---------------+
|        |--(A)-- Token Request ------------------->|               |
| Client |        (DPoP Proof)                      | Authorization |
|        |                                          |     Server    |
|        |<-(B)-- DPoP-bound Access Token ----------|               |
|        |        (token_type=DPoP)                 +---------------+
|        |
|        |
|        |                                          +---------------+
|        |--(C)-- DPoP-bound Access Token --------->|               |
|        |        (DPoP Proof)                      |    Resource   |
|        |                                          |     Server    |
|        |<-(D)-- Protected Resource ---------------|               |
|        |                                          +---------------+
+--------+
```

Figure 1: Basic DPoP Flow

The basic steps of an OAuth flow with DPoP are shown in Figure 1:

* (A) In the Token Request, the client sends an authorization grant (e.g., an authorization code, refresh token, etc.) to the authorization server in order to obtain an access token (and potentially a refresh token). The client attaches a DPoP proof to the request in an HTTP header.

* (B) The authorization server binds (sender-constrains) the access token to the public key claimed by the client in the DPoP proof; that is, the access token cannot be used without proving

possession of the respective private key. If a refresh token is issued to a public client, it too is bound to the public key of the DPoP proof.

* (C) To use the access token the client has to prove possession of the private key by, again, adding a header to the request that carries the DPoP proof. The resource server needs to receive information about the public key to which the access token is bound. This information may be encoded directly into the access token (for JWT structured access tokens) or provided via token introspection endpoint (not shown). The resource server verifies that the public key to which the access token is bound matches the public key of the DPoP proof.

* (D) The resource server refuses to serve the request if the signature check fails or the data in the DPoP proof is wrong, e.g., the request URI does not match the URI claim in the DPoP proof JWT. The access token itself, of course, must also be valid in all other respects.

The DPoP mechanism presented herein is not a client authentication method. In fact, a primary use case of DPoP is for public clients (e.g., single page applications and native applications) that do not use client authentication. Nonetheless, DPoP is designed such that it is compatible with private_key_jwt and all other client authentication methods.

DPoP does not directly ensure message integrity but relies on the TLS layer for that purpose. See Section 8 for details.

4. DPoP Proof JWTs

DPoP introduces the concept of a DPoP proof, which is a JWT created by the client and sent with an HTTP request using the DPoP header field. A valid DPoP proof demonstrates to the server that the client holds the private key that was used to sign the JWT. This enables authorization servers to bind issued tokens to the corresponding public key (as described in Section 5) and for resource servers to verify the key-binding of tokens that it receives (see Section 7.1), which prevents said tokens from being used by any entity that does not have access to the private key.

The DPoP proof demonstrates possession of a key and, by itself, is not an authentication or access control mechanism. When presented in conjunction with a key-bound access token as described in Section 7.1, the DPoP proof provides additional assurance about the legitimacy of the client to present the access token. But a valid DPoP proof JWT is not sufficient alone to make access control decisions.

4.1. The DPoP HTTP Header

A DPoP proof is included in an HTTP request using the following message header field.

DPoP A JWT that adheres to the structure and syntax of Section 4.2.

Figure 2 shows an example DPoP HTTP header field (line breaks and extra whitespace for display purposes only).


```
DPoP: eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0eSI6Ik
 VDIiwieCI6Imw4dEZyaHgtMzR0VjNoUklDUkRZOXpDa0RscEJoRjQyVVFVZldWQVdCR
 nMiLCJ5IjoiOVZFNGpmX09rX282NHpiVFRsY3VOSmFqSG10NnY5VERWclUwQ2R2R1JE
 QSIsImNydiI6IlAtMjU2In19.eyJqdGkiOiItQndDM0VTYzZhY2MybFRjIiwiaHRtIj
 oiUE9TVCIsImh0dSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29tL3Rva2VuIiwia
 WF0IjoxNTYyMjYyNjE2fQ.2-GxA6T8lP4vfrg8v-FdWP0A0zdrj8igiMLvqRMUvwnQg
 4PtFLbdLXiOSsX0x7NVY-FNyJK70nfbV37xRZT3Lg
```


Figure 2: Example DPoP header

Note that per [RFC7230] header field names are case-insensitive; so DPoP, DPOP, dpop, etc., are all valid and equivalent header field names. Case is significant in the header field value, however.

4.2. DPoP Proof JWT Syntax

A DPoP proof is a JWT ([RFC7519]) that is signed (using JWS, [RFC7515]) with a private key chosen by the client (see below). The header of a DPoP JWT contains at least the following parameters:

* typ: type header, value dpop+jwt (REQUIRED).

* alg: a digital signature algorithm identifier as per [RFC7518] (REQUIRED). MUST NOT be none or an identifier for a symmetric algorithm (MAC).

* jwk: representing the public key chosen by the client, in JWK format, as defined in [RFC7515] (REQUIRED)

The body of a DPoP proof contains at least the following claims:

* jti: Unique identifier for the DPoP proof JWT (REQUIRED). The value MUST be assigned such that there is a negligible probability that the same value will be assigned to any other DPoP proof used in the same context during the time window of validity. Such uniqueness can be accomplished by encoding (base64url or any other suitable encoding) at least 96 bits of pseudorandom data or by using a version 4 UUID string according

to [RFC4122]. The jti can be used by the server for replay detection and prevention, see Section 8.1.

* htm: The HTTP method for the request to which the JWT is attached, as defined in [RFC7231] (REQUIRED).

* htu: The HTTP URI used for the request, without query and fragment parts (REQUIRED).

* iat: Time at which the JWT was created (REQUIRED).

Figure 3 is a conceptual example showing the decoded content of the DPoP proof in Figure 2. The JSON of the JOSE header and payload are shown but the signature part is omitted. As usual, line breaks and extra whitespace are included for formatting and readability.


```
{
  "typ":"dpop+jwt",
  "alg":"ES256",
  "jwk": {
    "kty":"EC",
    "x":"l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
    "y":"9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
    "crv":"P-256"
  }
}
.
{
  "jti":"-BwC3ESc6acc2lTc",
  "htm":"POST",
  "htu":"https://server.example.com/token",
  "iat":1562262616
}
```


Figure 3: Example JWT content of a DPoP proof

Of the HTTP content in the request, only the HTTP method and URI are included in the DPoP JWT, and therefore only these 2 headers of the request are covered by the DPoP proof and its signature. The idea is sign just enough of the HTTP data to provide reasonable proof-of-possession with respect to the HTTP request. But that it be a minimal subset of the HTTP data so as to avoid the substantial difficulties inherent in attempting to normalize HTTP messages. Nonetheless, DPoP proofs can be extended to contain other information of the HTTP request (see also Section 8.4).

4.3. Checking DPoP Proofs

To check if a string that was received as part of an HTTP Request is a valid DPoP proof, the receiving server MUST ensure that

1. the string value is a well-formed JWT,

2. all required claims are contained in the JWT,

3. the typ field in the header has the value dpop+jwt,

4. the algorithm in the header of the JWT indicates an asymmetric digital signature algorithm, is not none, is supported by the application, and is deemed secure,

5. that the JWT is signed using the public key contained in the jwk header of the JWT,

6. the htm claim matches the HTTP method value of the HTTP request in which the JWT was received,

7. the htu claims matches the HTTPS URI value for the HTTP request in which the JWT was received, ignoring any query and fragment parts,

8. the token was issued within an acceptable timeframe (see Section 8.1), and

9. that, within a reasonable consideration of accuracy and resource utilization, a JWT with the same jti value has not previously been received at the same URI (see Section 8.1).

Servers SHOULD employ Syntax-Based Normalization and Scheme-Based Normalization in accordance with Section 6.2.2. and Section 6.2.3. of [RFC3986] before comparing the htu claim.

5. DPoP Access Token Request

To request an access token that is bound to a public key using DPoP, the client MUST provide a valid DPoP proof JWT in a DPoP header when making an access token request to the authorization server's token endpoint. This is applicable for all access token requests regardless of grant type (including, for example, the common authorization_code and refresh_token grant types but also extension grants such as the JWT authorization grant [RFC7523]). The HTTPS request shown in Figure 4 illustrates an such an access token request using an an authorization code grant with a DPoP proof JWT in the DPoP header (extra line breaks and whitespace for display purposes only).

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
DPoP: eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0eSI6Ik
 VDIiwieCI6Imw4dEZyaHgtMzR0VjNoUklDUkRZOXpDa0RscEJoRjQyVVFVZldWQVdCR
 nMiLCJ5IjoiOVZFNGpmX09rX282NHpiVFRsY3VOSmFqSG10NnY5VERWclUwQ2R2R1JE
 QSIsImNydiI6IlAtMjU2In19.eyJqdGkiOiItQndDM0VTYzZhY2MybFRjIiwiaHRtIj
 oiUE9TVCIsImh0dSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29tL3Rva2VuIiwia
 WF0IjoxNTYyMjYyNjE2fQ.2-GxA6T8lP4vfrg8v-FdWP0A0zdrj8igiMLvqRMUvwnQg
 4PtFLbdLXiOSsX0x7NVY-FNyJK70nfbV37xRZT3Lg

grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&code_verifier=bEaL42izcC-o-xBk0K2vuJ6U-y1p9r_wW2dFWIWgjz-
```

Figure 4: Token Request for a DPoP sender-constrained token using an authorization code

The DPoP HTTP header MUST contain a valid DPoP proof JWT. If the DPoP proof is invalid, the authorization server issues an error response per Section 5.2 of [RFC6749] with invalid_dpop_proof as the value of the error parameter.

To sender-constrain the access token, after checking the validity of the DPoP proof, the authorization server associates the issued access token with the public key from the DPoP proof, which can be accomplished as described in Section 6. A token_type of DPoP in the access token response signals to the client that the access token was bound to its DPoP key and can used as described in Section 7.1. The example response shown in Figure 5 illustrates such a response.

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

{
 "access_token": "Kz~8mXK1EalYznwH-LC-1fBAo.4Ljp~zsPE_NeO.gxU",
 "token_type": "DPoP",
 "expires_in": 2677,
 "refresh_token": "Q..Zkm29lexi8VnWg2zPW1x-tgGad0Ibc3s3EwM_Ni4-g"
}
```

Figure 5: Access Token Response

The example response in Figure 5 included a refresh token, which the client can use to obtain a new access token when the the previous one expires. Refreshing an access token is a token request using the refresh_token grant type made to the the authorization server's token endpoint. As with all access token requests, the client makes it a DPoP request by including a DPoP proof, which is shown in the Figure 6 example (extra line breaks and whitespace for display purposes only).

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
DPoP: eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0eSI6Ik
 VDIiwieCI6Imw4dEZyaHgtMzR0VjNoUklDUkRZOXpDa0RscEJoRjQyVVFVZldWQVdCR
 nMiLCJ5IjoiOVZFNGpmX09rX282NHpiVFRsY3VOSmFqSG10NnY5VERWclUwQ2R2R1JE
 QSIsImNydiI6IlAtMjU2In19.eyJqdGkiOiItQndDM0VTYzZhY2MybFRjIiwiaHRtIj
 oiUE9TVCIsImh0dSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29tL3Rva2VuIiwia
 WF0IjoxNTYyMjY1Mjk2fQ.pAqut2IRDm_De6PR93SYmGBPXpwrAk90e8cP2hjiaG5Qs
 GSuKDYW7_X620BxqhvYC8ynrrvZLTk41mSRroapUA

grant_type=refresh_token
&refresh_token=Q..Zkm29lexi8VnWg2zPW1x-tgGad0Ibc3s3EwM_Ni4-g
```

Figure 6: Token Request for a DPoP-bound token using a refresh token

When an authorization server supporting DPoP issues a refresh token to a public client that presents a valid DPoP proof at the token endpoint, the refresh token MUST be bound to the respective public key. The binding MUST be validated when the refresh token is later presented to get new access tokens. As a result, such a client MUST present a DPoP proof for the same key that was used to obtain the refresh token each time that refresh token is used to obtain a new access token. The implementation details of the binding of the refresh token are at the discretion of the authorization server. The server both produces and validates the refresh tokens that it issues so there's no interoperability consideration in the specific details of the binding.

An authorization server MAY elect to issue access tokens which are not DPoP bound, which is signaled to the client with a value of Bearer in the token_type parameter of the access token response per [RFC6750]. For a public client that is also issued a refresh token, this has the effect of DPoP-binding the refresh token alone, which can improve the security posture even when protected resources are not updated to support DPoP.

Refresh tokens issued to confidential clients (those having established authentication credentials with the authorization server) are not bound to the DPoP proof public key because they are already sender-constrained with a different existing mechanism. The OAuth 2.0 Authorization Framework [RFC6749] already requires that an authorization server bind refresh tokens to the client to which they were issued and that confidential clients authenticate to the authorization server when presenting a refresh token. As a result, such refresh tokens are sender-constrained by way of the client ID and the associated authentication requirement. This existing sender-constraining mechanism is more flexible (e.g., it allows credential rotation for the client without invalidating refresh tokens) than binding directly to a particular public key.

5.1. Authorization Server Metadata

This document introduces the following new authorization server metadata [RFC8414] parameter to signal support for DPoP in general and the specific JWS alg values the authorization server supports for DPoP proof JWTs.

dpop_signing_alg_values_supported A JSON array containing a list of the JWS alg values supported by the authorization server for DPoP proof JWTs.

6. Public Key Confirmation

Resource servers MUST be able to reliably identify whether an access token is bound using DPoP and ascertain sufficient information about the public key to which the token is bound in order to verify the binding with respect to the the presented DPoP proof (see Section 7.1). Such a binding is accomplished by associating the public key with the token in a way that can be accessed by the protected resource, such as embedding the JWK hash in the issued access token directly, using the syntax described in Section 6.1, or through token introspection as described in Section 6.2. Other methods of associating a public key with an access token are possible, per agreement by the authorization server and the protected resource, but are beyond the scope of this specification.

Resource servers supporting DPoP MUST ensure that the the public key from the DPoP proof matches the pubic key to which the access token is bound.

6.1. JWK Thumbprint Confirmation Method

When access tokens are represented as JSON Web Tokens (JWT) [RFC7519], the public key information SHOULD be represented using the jkt confirmation method member defined herein. To convey the hash of a public key in a JWT, this specification introduces the following new JWT Confirmation Method [RFC7800] member for use under the cnf claim.

jkt JWK SHA-256 Thumbprint Confirmation Method. The value of the jkt member MUST be the base64url encoding (as defined in [RFC7515]) of the JWK SHA-256 Thumbprint (according to [RFC7638]) of the DPoP public key (in JWK format) to which the access token is bound.

The following example JWT in Figure 7 with decoded JWT payload shown in Figure 8 contains a cnf claim with the jkt JWK thumbprint confirmation method member. The jkt value in these examples is the hash of the public key from the DPoP proofs in the examples in Section 5.


```
eyJhbGciOiJFUzI1NiIsImtpZCI6IkJlQUxrYiJ9.eyJzdWIiOiJzb21lb25lQGV4YW1
wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJuYmYiOjE
1NjIyNjI2MTEsImV4cCI6MTU2MjI2NjIxNiwiY25mIjp7ImprdCI6IjBaY09DT1JaTll
5LURXcHFxMzBqWnlKR0hUTjBkMkhnbEJWM3VpZ3VBNEkifX0.3Tyo8VTcn6u_PboUmAO
YUY1kfAavomW_YwYMkmRNizLJoQzWy2fCo79Zi5yObpIzjWb5xW4OGld7ESZrh0fsrA
```


Figure 7: JWT containing a JWK SHA-256 Thumbprint Confirmation


```
{
  "sub":"someone@example.com",
  "iss":"https://server.example.com",
  "nbf":1562262611,
  "exp":1562266216,
  "cnf":{"jkt":"0ZcOCORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I"}
}
```


Figure 8: JWT Claims Set with a JWK SHA-256 Thumbprint Confirmation

6.2. JWK Thumbprint Confirmation Method in Token Introspection

OAuth 2.0 Token Introspection [RFC7662] defines a method for a protected resource to query an authorization server about the active state of an access token as well as to determine metainformation about the token.

For a DPoP-bound access token, the hash of the public key to which the token is bound is conveyed to the protected resource as metainformation in a token introspection response. The hash is conveyed using the same cnf content with jkt member structure as the JWK thumbprint confirmation method, described in Section 6.1, as a top-level member of the introspection response JSON. Note that the resource server does not send a DPoP proof with the introspection request and the authorization server does not validate an access token's DPoP binding at the introspection endpoint. Rather the resource server uses the data of the introspection response to validate the access token binding itself locally.

The example introspection request in Figure 9 and corresponding response in Figure 10 illustrate an introspection exchange for the example DPoP-bound access token that was issued in Figure 5.

```
POST /as/introspect.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic cnM6cnM6TWt1LTZnX2xDektJZHo0ZnNON2tZY3lhK1Rp

token=Kz~8mXK1EalYznwH-LC-1fBAo.4Ljp~zsPE_NeO.gxU
```

Figure 9: Example Introspection Request

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

{
  "active": true,
  "sub": "someone@example.com",
  "iss": "https://server.example.com",
  "nbf": 1562262611,
  "exp": 1562266216,
  "cnf": {"jkt": "0ZcOCORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I"}
}
```

Figure 10: Example Introspection Response for a DPoP-Bound Access Token

7. Protected Resource Access

To make use of an access token that is bound to a public key using DPoP, a client MUST prove possession of the corresponding private key by providing a DPoP proof in the DPoP request header. As such, protected resource requests with a DPoP-bound access token necessarily must include both a DPoP proof as per Section 4 and the access token as described in Section 7.1.

7.1. The DPoP Authorization Request Header Scheme

A DPoP-bound access token is sent using the Authorization request header field per Section 2 of [RFC7235] using an authentication scheme of DPoP. The syntax of the Authorization header field for the DPoP scheme uses the token68 syntax defined in Section 2.1 of [RFC7235] (repeated below for ease of reference) for credentials. The Augmented Backus-Naur Form (ABNF) notation [RFC5234] syntax for DPoP Authorization scheme credentials is as follows:

```
token68    = 1*( ALPHA / DIGIT /
                 "-" / "." / "_" / "~" / "+" / "/" ) *"="

credentials = "DPoP" 1*SP token68
```

Figure 11: DPoP Authorization Scheme ABNF

For such an access token, a resource server MUST check that a DPoP proof was also received in the DPoP header field of the HTTP request, check the DPoP proof according to the rules in Section 4.3, and check that the public key of the DPoP proof matches the public key to which the access token is bound per Section 6.

The resource server MUST NOT grant access to the resource unless all checks are successful.

Figure 12 shows an example request to a protected resource with a DPoP-bound access token in the Authorization header and the DPoP proof in the DPoP header (line breaks and extra whitespace for display purposes only).

```
GET /protectedresource HTTP/1.1
Host: resource.example.org
Authorization: DPoP Kz~8mXK1EalYznwH-LC-1fBAo.4Ljp~zsPE_NeO.gxU
DPoP: eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0eSI6Ik
 VDIiwieCI6Imw4dEZyaHgtMzR0VjNoUklDUkRZOXpDa0RscEJoRjQyVVFVZldWQVdCR
 nMiLCJ5IjoiOVZFNGpmX09rX282NHpiVFRsY3VOSmFqSG10NnY5VERWclUwQ2R2R1JE
 QSIsImNydiI6IlAtMjU2In19.eyJqdGkiOiJlMWozVl9iS2ljOC1MQUVCIiwiaHRtIj
 oiR0VUIiwiaHR1IjoiaHR0cHM6Ly9yZXNvdXJjZS5leGFtcGxlLm9yZy9wcm90ZWN0Z
 WRyZXNvdXJjZSIsImlhdCI6MTU2MjI2MjYxOH0.lNhmpAX1WwmpBvwhok4E74kWCiGB
 NdavjLAeevGy32H3dbF0Jbri69Nm2ukkwb-uyUI4AUg1JSskfWIyo4UCbQ
```

Figure 12: DPoP Protected Resource Request

Upon receipt of a request for a URI of a protected resource within the protection space requiring DPoP authorization, if the request does not include valid credentials or does not contain an access token sufficient for access to the protected resource, the server can reply with a challenge using the 401 (Unauthorized) status code ([RFC7235], Section 3.1) and the WWW-Authenticate header field ([RFC7235], Section 4.1). The server MAY include the WWW-Authenticate header in response to other conditions as well.

In such challenges:

* The scheme name is DPoP.

* The authentication parameter realm MAY be included to indicate the scope of protection in the manner described in [RFC7235], Section 2.2.

* A scope authentication parameter MAY be included as defined in [RFC6750], Section 3.

* An error parameter ([RFC6750], Section 3) SHOULD be included to indicate the reason why the request was declined, if the request included an access token but failed authorization. Parameter values are described in Section 3.1 of [RFC6750].

* An error_description parameter ([RFC6750], Section 3) MAY be included along with the error parameter to provide developers a human-readable explanation that is not meant to be displayed to end-users.

* An algs parameter SHOULD be included to signal to the client the JWS algorithms that are acceptable for the DPoP proof JWT. The value of the parameter is a space-delimited list of JWS alg (Algorithm) header values ([RFC7515], Section 4.1.1).

* Additional authentication parameters MAY be used and unknown parameters MUST be ignored by recipients

For example, in response to a protected resource request without authentication:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: DPoP realm="WallyWorld", algs="ES256 PS256"
```

Figure 13: HTTP 401 Response To A Protected Resource Request Without Authentication

And in response to a protected resource request that was rejected because the confirmation of the DPoP binding in the access token failed:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: DPoP realm="WallyWorld", error="invalid_token",
  error_description="Invalid DPoP key binding", algs="ES256"
```

Figure 14: HTTP 401 Response To A Protected Resource Request With An Invalid Token

7.2. The Bearer Authorization Request Header Scheme

Protected resources simultaneously supporting both the DPoP and Bearer schemes need to update how evaluation of bearer tokens is performed to prevent downgraded usage of a DPoP-bound access tokens. Specifically, such a protected resource MUST reject an access token received as a bearer token per [!@RFC6750], if that token is determined to be DPoP-bound.

A protected resource that supports only [RFC6750] and is unaware of DPoP would most presumably accept a DPoP-bound access token as a bearer token (JWT [RFC7519] says to ignore unrecognized claims, Introspection [RFC7662] says that other parameters might be present while placing no functional requirements on their presence, and [RFC6750] is effectively silent on the content of the access token as it relates to validity). As such, a client MAY send a DPoP-bound access token using the Bearer scheme upon receipt of a WWW-Authenticate: Bearer challenge from a protected resource (or if it has prior such knowledge about the capabilities of the protected resource). The effect of this likely simplifies the logistics of phased upgrades to protected resources in their support DPoP or even prolonged deployments of protected resources with mixed token type support.

8. Security Considerations

In DPoP, the prevention of token replay at a different endpoint (see Section 2) is achieved through the binding of the DPoP proof to a certain URI and HTTP method. DPoP, however, has a somewhat different nature of protection than TLS-based methods such as OAuth Mutual TLS [RFC8705] or OAuth Token Binding [I-D.ietf-oauth-token-binding] (see also Section 8.1 and Section 8.4). TLS-based mechanisms can leverage a tight integration between the TLS layer and the application layer to achieve a very high level of message integrity with respect to the transport layer to which the token is bound and replay protection in general.

8.1. DPoP Proof Replay

If an adversary is able to get hold of a DPoP proof JWT, the adversary could replay that token at the same endpoint (the HTTP endpoint and method are enforced via the respective claims in the JWTs). To prevent this, servers MUST only accept DPoP proofs for a limited time window after their iat time, preferably only for a relatively brief period. Servers SHOULD store, in the context of the request URI, the jti value of each DPoP proof for the time window in which the respective DPoP proof JWT would be accepted and decline HTTP requests to the same URI for which the jti value has been seen before. In order to guard against memory exhaustion attacks a server SHOULD reject DPoP proof JWTs with unnecessarily large jti values or store only a hash thereof.

Note: To accommodate for clock offsets, the server MAY accept DPoP proofs that carry an iat time in the reasonably near future (e.g., a few seconds in the future).

8.2. Signed JWT Swapping

Servers accepting signed DPoP proof JWTs MUST check the typ field in the headers of the JWTs to ensure that adversaries cannot use JWTs created for other purposes.

8.3. Signature Algorithms

Implementers MUST ensure that only asymmetric digital signature algorithms that are deemed secure can be used for signing DPoP proofs. In particular, the algorithm none MUST NOT be allowed.

8.4. Message Integrity

DPoP does not ensure the integrity of the payload or headers of requests. The DPoP proof only contains claims for the HTTP URI and method, but not, for example, the message body or general request headers.

This is an intentional design decision intended to keep DPoP simple to use, but as described, makes DPoP potentially susceptible to replay attacks where an attacker is able to modify message contents and headers. In many setups, the message integrity and confidentiality provided by TLS is sufficient to provide a good level of protection.

Implementers that have stronger requirements on the integrity of messages are encouraged to either use TLS-based mechanisms or signed requests. TLS-based mechanisms are in particular OAuth Mutual TLS [RFC8705] and OAuth Token Binding [I-D.ietf-oauth-token-binding].

Note: While signatures covering other parts of requests are out of the scope of this specification, additional information to be signed can be added into DPoP proofs.

8.5. Public Key Binding

The binding between the DPoP public key and the access token, which is specified in Section 6, uses a cryptographic hash of the JWK representation of the public key. It relies on the hash function having sufficient second-preimage resistance so as to make it computationally infeasible to find or create another key that produces to the same hash output value. The SHA-256 hash function was used because it meets the aforementioned requirement while being widely available. If, in the future, JWK thumbprints need to be computed using hash function(s) other than SHA-256, it is suggested that, for additional related JWT confirmation methods, members be defined for that purpose and registered in the IANA "JWT Confirmation Methods" registry [IANA.JWT.Claims] for JWT "cnf" member values.

9. IANA Considerations

9.1. OAuth Access Token Type Registration

This specification requests registration of the following access token type in the "OAuth Access Token Types" registry [IANA.OAuth.Params] established by [RFC6749].

* Type name: DPoP
* Additional Token Endpoint Response Parameters: (none)
* HTTP Authentication Scheme(s): DPoP
* Change controller: IESG
* Specification document(s): [[this specification]]

9.2. HTTP Authentication Scheme Registration

This specification requests registration of the following scheme in the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" [RFC7235][IANA.HTTP.AuthSchemes]:

* Authentication Scheme Name: DPoP
* Reference: [[Section 7.1 of this specification]]

9.3. Media Type Registration

[[Is a media type registration at [IANA.MediaTypes] necessary for application/dpop+jwt? There is a +jwt structured syntax suffix registered already at [IANA.MediaType.StructuredSuffix] by Section 7.2 of [RFC8417], which is maybe sufficient? A full-blown registration of application/dpop+jwt seems like it'd be overkill. The dpop+jwt is used in the JWS/JWT typ header for explicit typing of the JWT per Section 3.11 of [RFC8725] but it is not used anywhere else (such as the Content-Type of HTTP messages).

Note that there does seem to be some precedence for [IANA.MediaTypes] registration with [I-D.ietf-oauth-access-token-jwt], [I-D.ietf-oauth-jwsreq], [RFC8417], and of course [RFC7519]. But precedence isn't always right.]]

9.4. JWT Confirmation Methods Registration

This specification requests registration of the following value in the IANA "JWT Confirmation Methods" registry [IANA.JWT] for JWT cnf member values established by [RFC7800].

* Confirmation Method Value: jkt
* Confirmation Method Description: JWK SHA-256 Thumbprint
* Change Controller: IESG
* Specification Document(s): [[Section 6 of this specification]]

9.5. JSON Web Token Claims Registration

This specification requests registration of the following Claims in the IANA "JSON Web Token Claims" registry [IANA.JWT] established by [RFC7519].

HTTP method:

* Claim Name: htm
* Claim Description: The HTTP method of the request
* Change Controller: IESG
* Specification Document(s): [[Section 4.2 of this specification]]

HTTP URI:

* Claim Name: htu
* Claim Description: The HTTP URI of the request (without query and fragment parts)
* Change Controller: IESG
* Specification Document(s): [[Section 4.2 of this specification]]

9.6. HTTP Message Header Field Names Registration

This document specifies the following new HTTP header fields, registration of which is requested in the "Permanent Message Header Field Names" registry [IANA.Headers] defined in [RFC3864].

* Header Field Name: DPoP

*Applicable protocol: HTTP

*Status: standard

*Author/change Controller: IETF

*Specification Document(s): [[this specification]]

9.7. Authorization Server Metadata Registration

This specification requests registration of the following values in the IANA "OAuth Authorization Server Metadata" registry [IANA.OAuth.Parameters] established by [RFC8414].

*Metadata Name: dpop_signing_alg_values_supported

*Metadata Description: JSON array containing a list of the JWS algorithms supported for DPoP proof JWTs

*Change Controller: IESG

*Specification Document(s): [[Section 5.1 of this specification]]

10. Normative References

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

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

[RFC7638] 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>.

[RFC5234] 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>.

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

[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/info/rfc7231>.

[RFC3986] 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>.

[RFC7800] 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. Informative References

[I-D.ietf-oauth-security-topics]
Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "OAuth 2.0 Security Best Current Practice", Work in Progress, Internet-Draft, draft-ietf-oauth-security-topics-16, 5 October 2020, <https://tools.ietf.org/html/draft-ietf-oauth-security-topics-16>.

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

[IANA.OAuth.Params] IANA, "OAuth Parameters", <https://www.iana.org/assignments/oauth-parameters>.

[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/info/rfc8725>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/info/rfc8414>.

[I-D.ietf-oauth-access-token-jwt]
Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", Work in Progress, Internet-Draft, draft-ietf-oauth-access-token-jwt-10, 23 September 2020, <https://tools.ietf.org/html/draft-ietf-oauth-access-token-jwt-10>.

[IANA.Headers]
IANA, "Message Headers", <https://www.iana.org/assignments/message-headers>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[W3C.WebCryptoAPI] Watson, M., "Web Cryptography API", 26 January 2017, <https://www.w3.org/TR/2017/REC-WebCryptoAPI-20170126>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/info/rfc8705>.

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, <https://www.rfc-editor.org/info/rfc8707>.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <https://www.rfc-editor.org/info/rfc7523>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/info/rfc7235>.

[IANA.JWT] IANA, "JSON Web Token Claims", <http://www.iana.org/assignments/jwt>.

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

[IANA.HTTP.AuthSchemes] IANA, "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry", <https://www.iana.org/assignments/http-authschemes>.

[IANA.MediaType.StructuredSuffix] IANA, "Structured Syntax Suffix Registry", <https://www.iana.org/assignments/media-type-structured-suffix>.

[RFC8417] Hunt, P., Ed., Jones, M., Denniss, W., and M. Ansari, "Security Event Token (SET)", RFC 8417, DOI 10.17487/RFC8417, July 2018, <https://www.rfc-editor.org/info/rfc8417>.

[I-D.ietf-oauth-token-binding]
Jones, M., Campbell, B., Bradley, J., and W. Denniss, "OAuth 2.0 Token Binding", Work in Progress, Internet-Draft, draft-ietf-oauth-token-binding-08, 19 October 2018, <https://tools.ietf.org/html/draft-ietf-oauth-token-binding-08>.

[RFC6750] 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>.

[IANA.MediaTypes] IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

[I-D.ietf-oauth-jwsreq]
Sakimura, N., Bradley, J., and M. Jones, "The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request (JAR)", Work in Progress, Internet-Draft, draft-ietf-oauth-jwsreq-30, 10 September 2020, <https://tools.ietf.org/html/draft-ietf-oauth-jwsreq-30>.

Appendix A. Acknowledgements

We would like to thank Annabelle Backman, Dominick Baier, William Denniss, Vladimir Dzhuvinov, Mike Engan, Nikos Fotiou, Mark Haine, Dick Hardt, Bjorn Hjelm, Jared Jennings, Steinar Noem, Neil Madden, Rob Otto, Aaron Parecki, Michael Peck, Paul Querna, Justin Richer, Filip Skokan, Dave Tonge, Jim Willeke, and others (please let us know, if you've been mistakenly omitted) for their valuable input, feedback and general support of this work.

This document resulted from discussions at the 4th OAuth Security Workshop in Stuttgart, Germany. We thank the organizers of this workshop (Ralf Kusters, Guido Schmitz).

Appendix B. Document History

[[To be removed from the final specification]]

-02

* Lots of editorial updates and additions including expanding on the objectives, better defining the key confirmation representations, example updates and additions, better describing mixed bearer/dpop token type deployments, clarify RT binding only being done for public clients and why, more clearly allow for a bound RT but with bearer AT, explain/justify the choice of SHA-256 for key binding, and more

* Require that a protected resource supporting bearer and DPoP at the same time must reject an access token received as bearer, if that token is DPoP-bound

* Remove the case-insensitive qualification on the htm claim check

* Relax the jti tracking requirements a bit and qualify it by URI

-01

* Editorial updates

* Attempt to more formally define the DPoP Authorization header scheme

* Define the 401/WWW-Authenticate challenge

* Added invalid_dpop_proof error code for DPoP errors in token request

* Fixed up and added to the IANA section

* Added dpop_signing_alg_values_supported authorization server metadata

* Moved the Acknowledgements into an Appendix and added a bunch of names (best effort)

-00 [[Working Group Draft]]

* Working group draft

-04

*Update OAuth MTLS reference to RFC 8705

*Use the newish RFC v3 XML and HTML format

-03

*rework the text around uniqueness requirements on the jti claim in the DPoP proof JWT

*make tokens a bit smaller by using htm, htu, and jkt rather than http_method, http_uri, and jkt#S256 respectively

*more explicit recommendation to use mTLS if that is available

*added David Waite as co-author

*editorial updates

-02

*added normalization rules for URIs

*removed distinction between proof and binding

*"jwk" header again used instead of "cnf" claim in DPoP proof

*renamed "Bearer-DPoP" token type to "DPoP"

*removed ability for key rotation

*added security considerations on request integrity

*explicit advice on extending DPoP proofs to sign other parts of the HTTP messages

*only use the jkt#S256 in ATs

*iat instead of exp in DPoP proof JWTs

*updated guidance on token_type evaluation

-01

*fixed inconsistencies

*moved binding and proof messages to headers instead of parameters

*extracted and unified definition of DPoP JWTs *improved description

-00

*first draft

Authors' Addresses

Daniel Fett
yes.com

Email: mail@danielfett.de

Brian Campbell
Ping Identity

Email: bcampbell@pingidentity.com

John Bradley
Yubico

Email: ve7jtb@ve7jtb.com

Torsten Lodderstedt
yes.com

Email: torsten@lodderstedt.net

Michael Jones
Microsoft

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

David Waite
Ping Identity

Email: david@alkaline-solutions.com