

Workgroup: Web Authorization Protocol
Internet-Draft: draft-fett-oauth-dpop-04
Published: 4 March 2020

Intended Status: Standards Track

Expires: 5 September 2020

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 Demonstration of 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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 5 September 2020.

Copyright Notice

Copyright (c) 2020 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 (<https://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](#)
 - [1.1. Conventions and Terminology](#)
- [2. Main Objective](#)
- [3. Concept](#)
- [4. DPoP Proof JWTs](#)
 - [4.1. Syntax](#)
 - [4.2. Checking DPoP Proofs](#)
- [5. Token Request \(Binding Tokens to a Public Key\)](#)
- [6. Resource Access \(Proof of Possession for Access Tokens\)](#)
- [7. Public Key Confirmation](#)
- [8. Acknowledgements](#)
- [9. Security Considerations](#)
 - [9.1. DPoP Proof Replay](#)
 - [9.2. Signed JWT Swapping](#)
 - [9.3. Signature Algorithms](#)
 - [9.4. Message Integrity](#)
- [10. IANA Considerations](#)
 - [10.1. OAuth Access Token Type Registration](#)
 - [10.2. JSON Web Signature and Encryption Type Values Registration](#)
- [11. Normative References](#)
- [12. Informative References](#)
- [Appendix A. Document History](#)
- [Authors' Addresses](#)

1. Introduction

[[RFC8705](#)] describes methods to bind (sender-constrain) access tokens using mutual Transport Layer Security (TLS) authentication with X.509 certificates.

[[I-D.ietf-oauth-token-binding](#)] provides mechanisms to sender-constrain access tokens using HTTP token binding.

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.

This document outlines an application-level sender-constraining for access and refresh tokens that can be used in cases where neither mTLS nor OAuth Token Binding are available. It uses proof-of-possession based on a public/private key pair and application-level signing.

DPoP can be used with public clients and, in case of confidential clients, can be combined with any client authentication method.

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. Main Objective

Under the attacker model defined in [[I-D.ietf-oauth-security-topics](#)], the mechanism defined by this specification aims to prevent token replay at a different endpoint.

More precisely, if an adversary is able to get hold of an access token or refresh token because it set up a counterfeit authorization server or resource server, the adversary is not able to replay the respective token at another authorization or resource server.

Secondary objectives are discussed in [Section 9](#).

3. Concept

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



Figure 1

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 code 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 AS 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. This is signaled to the client by using the token_type value DPoP.
- *If a refresh token is issued to a public client, it is sender-constrained in the same way. For confidential clients, refresh tokens are bound to the client_id, which is more flexible than binding it to a particular public key.
- *(C) If the client wants to use the access token, it 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 is either encoded directly into the access token (for JWT structured access tokens), or provided at the token introspection endpoint of the authorization server (not shown).

*(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.

*When a refresh token that is sender-constrained using DPoP is used by the client, the client has to provide a DPoP proof just as in the case of a resource access. The new access token will be bound to the same public key.

The mechanism presented herein is not a client authentication method. In fact, a primary use case is public clients (single page 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 9](#) for details.

4. DPoP Proof JWTs

DPoP uses so-called DPoP proof JWTs for binding public keys and proving knowledge about private keys.

4.1. Syntax

A DPoP proof is a JWT ([\[RFC7519\]](#)) that is signed (using JWS, [\[RFC7515\]](#)) using 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 SHOULD be used by the server for replay detection and prevention, see [Section 9.1](#).

*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 2 shows the JSON header and payload of a DPoP proof JWT.

```
{
  "typ": "dpop+jwt",
  "alg": "ES256",
  "jwk": {
    "kty": "EC",
    "x": "18tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
    "y": "9VE4jf_0k_o64zbTT1cuNJajHmt6v9TDVrU0CdvGRDA",
    "crv": "P-256"
  }
}.{
  "jti": "-BwC3ESc6acc2lTc",
  "htm": "POST",
  "htu": "https://server.example.com/token",
  "iat": 1562262616
}
```

Figure 2

Figure 2: Example JWT content for DPoP proof header.

Note: To keep DPoP simple to implement, only the HTTP method and URI are signed in DPoP proofs. Nonetheless, DPoP proofs can be extended to contain other information of the HTTP request (see also [Section 9.4](#)).

4.2. 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 (case-insensitive),
7. the htu claims matches the HTTP 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 9.1](#)), and
9. that, within a reasonable consideration of accuracy and resource utilization, a JWT with the same jti value has not been received previously (see [Section 9.1](#)).

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. Token Request (Binding Tokens to a Public Key)

To bind a token to a public key in the token request, the client MUST provide a valid DPoP proof JWT in a DPoP header. The HTTPS request shown in Figure 3 illustrates the protocol for this (with extra line breaks for display purposes only).

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
DPoP: eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVMTjU2IiwiaWdrIjpw7Imt0eSI6Ik
VDIiwieCI6Imw4dEZyaHgtMzR0VjNoUk1DUkRZOXpDa0RscEJoRjQyVVFVZldwQVdCR
nMiLCJ5IjoioVZFNGpmX09rX282NHpiVFRsY3V0SmFqSG10NnY5VERWclUwQ2R2R1JE
QSIsImNydiI6IiAtMjU2In19.eyJqdGkiOiItQndDM0VTYzZHY2MybFRjIiwiaHRtIj
oiUE9TVCIsImh0dSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29tL3Rva2VuIiwia
WF0IjoxNTYyMjYyNjE2fQ.2-GxA6T8lP4vfrg8v-FdWP0A0zdrj8igiMLvqRMUvwnQg
4PtFLbdLXi0SsX0x7NVY-FNyJK70nfbv37xRZT3Lg
grant_type=authorization_code
&code=Splxl0BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&code_verifier=bEaL42izcC-o-xBk0K2vuJ6U-y1p9r_wW2dFWIWgjz-
```

Figure 3

Figure 3: Token Request for a DPoP sender-constrained token.

The HTTP header DPoP MUST contain a valid DPoP proof.

The authorization server, after checking the validity of the DPoP proof, MUST associate the access token issued at the token endpoint with the public key. It then sets token_type to DPoP in the token response.

A client typically cannot know whether a certain AS supports DPoP. It therefore SHOULD use the value of the token_type parameter returned from the AS to determine support for DPoP: If the token type returned is Bearer or another value, the AS does not support DPoP. If it is DPoP, DPoP is supported. Only then, the client needs to send the DPoP header in subsequent requests and use the token type DPoP in the Authorization header as described below.

If a refresh token is issued to a public client at the token endpoint and a valid DPoP proof is presented, the refresh token MUST be bound to the public key contained in the header of the DPoP proof JWT.

If a DPoP-bound refresh token is to be used at the token endpoint by a public client, the AS MUST ensure that the DPoP proof contains the same public key as the one the refresh token is bound to. The access token issued MUST be bound to the public key contained in the DPoP proof.

6. Resource Access (Proof of Possession for Access Tokens)

To make use of an access token that is token-bound to a public key using DPoP, a client MUST prove the possession of the corresponding private key by providing a DPoP proof in the DPoP request header.

The DPoP-bound access token must be sent in the Authorization header with the prefix DPoP.

If a resource server detects that an access token that is to be used for resource access is bound to a public key using DPoP (via the methods described in [Section 7](#)) it MUST check that a header DPoP was received in the HTTP request, and check the header's contents according to the rules in [Section 4.2](#).

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


```

GET /protectedresource HTTP/1.1
Host: resource.example.org
Authorization: DPoP eyJhbGciOiJFUzI1NiIsImtpZCI6IkJlQUxrYiJ9.eyJzdWIiOiJzb21lb25lQGV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJhdWQiOiJodHRwczovL3Jlc291cmNlLmV4YW1wbGUub3JnIiwibmJmIjoxNTYyMjYyYnJlcjE1NjIyYnJyMTYsImNuZiI6eyJqa3QiOiIwWmNPNQ09Szk5ZeS1EV3BxcTMwalp5SkdIVE4wZDJIZ2xCVjN1aWd1QTRJIn19.vsFiVqHCyIkBYu50c69bMPJsJ8qYlsXfuC6nZcLl8YRNOhqMuRXu6oSZHe2dGZY00DNaGg1cg-kVigzYhF1MQ
DPoP: eyJ0eXAiOiJkcG9wK2p3dCI6ImFsZyI6IkVMTjU2IiwiaWdrIjpw7Imt0eSI6IkVDIiwieCI6Imw4dEZyaHgtMzR0VjNoUk1DUkRZOXpDa0RscEJoRjQyVVFVZldWQvdCRnMiLCJ5IjoioVZFNzGmX09rX282NHpiVFRsY3V0SmFqSG10NnY5VERWclUwQ2R2R1JEQSIsImNydiI6IlAtMjU2In19.eyJqdGkiOiJlMwozVl9iS2ljOC1MQUVCIiwiaHRtIjoiR0VUIiwiaHR1IjoiaHR0cHM6Ly9yZXNvdXJzS5leGFtcGxlM9yZy9wcm90ZWNOZWRyZXNvdXJzSIsIm1hdCI6MTU2MjI2MjYxOH0.lNhmpAX1WwmpBvwhok4E74kWcIGBNdavyLAeevGy32H3dbF0Jbri69Nm2ukkbw-uyUI4AUg1JSskfWIyo4UCbQ

```

Figure 4

Figure 4: Protected Resource Request with a DPoP sender-constrained access token.

7. Public Key Confirmation

It MUST be ensured that resource servers can reliably identify whether a token is bound using DPoP and learn the public key to which the token is bound.

Access tokens that are represented as JSON Web Tokens (JWT) [[RFC7519](#)] MUST contain information about the DPoP public key (in JWK format) in the member `jkt` of the `cnf` claim, as shown in Figure 5.

The value in `jkt` MUST be the `base64url` encoding [[RFC7515](#)] of the JWK SHA-256 Thumbprint (according to [[RFC7638](#)]) of the public key to which the access token is bound.

```

{
  "sub": "someone@example.com",
  "iss": "https://server.example.com",
  "aud": "https://resource.example.org",
  "nbf": 1562262611,
  "exp": 1562266216,
  "cnf": {
    "jkt": "0Zc0CORZNYy-DWpqq30jZyJGHTN0d2Hg1BV3uiguA4I"
  }
}

```

Figure 5

Figure 5: Example access token body with `cnf` claim.

When access token introspection is used, the same cnf claim as above MUST be contained in the introspection response.

Resource servers MUST ensure that the fingerprint of the public key in the DPoP proof JWT equals the value in the jkt claim in the access token or introspection response.

8. Acknowledgements

We would like to thank David Waite, Filip Skokan, Mike Engan, and Justin Richer for their valuable input and feedback.

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).

9. 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 does not, however, achieve the same level of protection as TLS-based methods such as OAuth Mutual TLS [[RFC8705](#)] or OAuth Token Binding [[I-D.ietf-oauth-token-binding](#)] (see also [Section 9.1](#) and [Section 9.4](#)). 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 and replay protection. Therefore, it is RECOMMENDED to prefer TLS-based methods over DPoP if such methods are suitable for the scenario at hand.

9.1. 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 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 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 near future (e.g., up to a few seconds in the future).

9.2. 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 in the DPoP headers.

9.3. 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.

9.4. Message Integrity

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

This is an intentional design decision 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 on (parts of) requests are out of the scope of this specification, signatures or information to be signed can be added into DPoP proofs.

10. IANA Considerations

10.1. OAuth Access Token Type Registration

This specification registers the following access token type in the OAuth Access Token Types registry defined in [RFC6749].

*Type name: "DPoP"

*Additional Token Endpoint Response Parameters: (none)

*HTTP Authentication Scheme(s): Bearer

*Change controller: IETF

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

10.2. JSON Web Signature and Encryption Type Values Registration

This specification registers the dpop+jwt type value in the IANA JSON Web Signature and Encryption Type Values registry [[RFC7515](#)]:

*"typ" Header Parameter Value: "dpop+jwt"

*Abbreviation for MIME Type: None

*Change Controller: IETF

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

11. Normative References

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

12. Informative References

[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>>.

[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>>.

[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>>.

[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>>.

[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>>.

[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-14, 10 February 2020, <<https://tools.ietf.org/html/draft-ietf-oauth-security-topics-14>>.

Appendix A. Document History

[[To be removed from the final specification]]

-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

David Waite
Ping Identity

Email: david@alkaline-solutions.com