DPOp Shephard Review Review

https://mailarchive.ietf.org/arch/msg/oauth/c9ipOQuRzMHDWSrLBewLv4GYbqC/

IETF 114 Philadelphia
OAuth WG (side meeting)
July 2022
DPoP Proof has a hash of the Access Token

2. The DPoP Proof contains a hash of the Access Token, and the Access Token contains a hash of the public key in the DPoP Proof. Why do you need both? Would one of these be sufficient?

The latter (AT containing a hash of the public key in the DPoP Proof) is needed and largely sufficient for the main goals of binding the AT to a key held by the client. The former (DPoP Proof containing a hash of the AT) was added later via very rough WG consensus - it can prevent some esoteric swapping of tokens that I never really understood to be honest and also limits the impact of using maliciously precomputed and exfiltrated proofs (https://www.ietf.org/archive/id/draft-ietf-oauth-dpop-10.html#section-2.6 talks about it a bit). Use of the nonce mechanism, which was added to the draft even later, also (and better) protects against precomputed and exfiltrated proofs. The value of the AT hash in the proof seems somewhat questionable. To me anyway. But removing it at this point is potentially problematic due to inertia, existing implementations/deployments, rough WG consensus, and more.

I think that at least a text is needed to justify this, and explain the "it can prevent some esoteric swapping of tokens" issue. Maybe we can discuss this during one of the side meetings in Philly.

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. To prevent this, servers can optionally require clients to include a server-chosen nonce into the proof that cannot be predicted by an attacker (nonce). In the absence of the optional nonce, the impact of pre-computed DPoP proofs is limited somewhat by the proof being bound to an access token on protected resource access. Because a proof covering an access token that does not yet exist cannot feasibly be created, access tokens obtained with an exfiltrated refresh token and pre-computed proofs will be unusable.

https://www.ietf.org/archive/id/draft-ietf-oauth-dpop-10.html#section-2.6 (and various)
PAR, DPoP, and Authorization Code Binding

10.1. DPoP with Pushed Authorization Requests

When Pushed Authorization Requests (PAR, [RFC9126]) are used in conjunction with DPoP, there are two ways in which the DPoP key can be communicated in the PAR request:

- The dpop._jkt parameter can be used as described above to bind the issued authorization code to a specific key. In this case, dpop._jkt MUST be included alongside other authorization request parameters in the POST body of the PAR request.
- Alternatively, the DPoP header can be added to the PAR request. In this case, the authorization server MUST check the provided DPoP proof JWT as defined in Section 4.3. It MUST further behave as if the contained public key’s thumbprint was provided using dpop._jkt, i.e., reject the subsequent token request unless a DPoP proof for the same key is provided. This can help to simplify the implementation of the client, as it can "blindly" attach the DPoP header to all requests to the authorization server regardless of the type of request. Additionally, it provides a stronger binding, as the DPoP header contains a proof of possession of the private key.

Both mechanisms MUST be supported by an authorization server that supports PAR and DPoP. If both mechanisms are used at the same time, the authorization server MUST reject the request if the JWK Thumbprint in dpop._jkt does not match the public key in the DPoP header.

Section 10.1

Why define two different mechanisms to achieve the same thing?
This seems to add complexity without an obvious benefit.

This is a bit of a tricky area. The benefit with PAR is the direct request from client to AS, which allows for an actual DPoP proof to be used for the eventual binding of the authorization code to the key. Also the client doesn’t have to do the JWK hash in that case. Whereas the normal authorization request is indirect via the browser and just a hash of the key is given for the code binding with the dpop._jkt parameter. And the client has to compute the hash. But PAR is just an alternative way to pass the authorization parameters (like dpop._jkt) so it’s kinda awkward to use things together like this. [GitHub Issue 103](https://github.com/danielefteff/draft-dpop/issues/103) and [111](https://github.com/danielefteff/draft-dpop/pull/111) have some discussion around this but there was some in person talk too so that’s not complete.

I don’t love that there’s two different mechanisms here. But it’s what we were able to come up with given all the factors. Certainly open to considering improvements but am pretty much at a loss of what that might be.

Let’s discuss this during one of the side meetings in Philly

[Link](https://www.ietf.org/archive/id/draft-ietf-oauth-dpop-10.html#section-10.1)
JWS Asymmetric Digital Signature Algorithms

11.6. 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.

Section 11.6
Should the algorithms be explicitly called out? Or at least reference a document that calls out such algorithms?

There isn’t a single such document and it’s not necessarily a static list of algorithms. I was about to say we could point to the JOSE alg registry but glancing again at it [https://www.iana.org/assignments/jose/jose.xhtml#web-signature-encryption-algorithms](https://www.iana.org/assignments/jose/jose.xhtml#web-signature-encryption-algorithms) and I suspect that’d confuse more than help. We could perhaps list some/many of the algs with the qualification that it’s not an exclusive or complete list? But I’m not sure how useful that would be, to be honest.

If you do not specify any algorithm, how do you ensure interop?
I think this is worth a discussion in Philly

[https://www.ietf.org/archive/id/draft-ietf-oauth-dpop-10.html#name-signature-algorithms](https://www.ietf.org/archive/id/draft-ietf-oauth-dpop-10.html#name-signature-algorithms)
Algorithm Agility (hash)

11.8. Access Token and Public Key Binding

The binding of the access token to the DPOp public key, 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 Thumbsprints need to be computed using hash function(s) other than SHA-256, it is suggested that an additional related JWK confirmation method member be defined for that purpose, registered in the respective IANA registry, and used in place of the jkt confirmation method defined herein.

Similarly, the binding of the DPOp proof to the access token uses a hash of that access token as the value of the ath claim in the DPOp proof (see Section 4.2). This relies on the value of the hash being sufficiently unique so as to reliably identify the access token. The collision resistance of SHA-256 meets that requirement. If, in the future, access token digests need be computed using hash function(s) other than SHA-256, it is suggested that an additional related JWT claim be defined for that purpose, registered in the respective IANA registry, and used in place of the ath claim defined herein.

Section 11.8

Why not include algorithm agility to make sure the mechanism is ready to allow for more secure algorithms in the future?

Algorithm agility is a whole can of worms that can be accomplished in different ways with different amounts of added complexity and potential vulnerabilities and issues of interop and MTI. Section 11.8 describes how DPOp allows for algorithm agility (without using the exact words) by suggesting that new dpop binding cnf method and/or AT hash claim be defined using a “better” hash algorithm if/when the need arises (OAuth 2.0 Mutual-TLS takes a similar approach FWIW). The intent of doing it that way was to keep things as simple as possible in the spec right now without completely closing the door on future needs.

This is another topic that is worth a discussion in Philly.

https://www.ietf.org/archive/id/draft-ietf-oauth-dpop-10.html#name-access-token-and-public-key
Hopefully we aren’t talking about this in London (IETF 115)