DBSC/DPOP as Complementary Technologies to FIDO Authentication

May 2025

Editors:

Shane Weeden, IBM

An Ho, IBM

Abstract

Session hijacking is a growing initial attack vector for online fraud and account takeover. Because FIDO authentication reduces the effectiveness of other simpler forms of compromise, such as credential stuffing and phishing, cybercriminals turn to theft and re-use of bearer tokens. Bearer tokens are a form of credential which include session cookies used by browsers connecting to websites and OAuth access tokens used by other thick client application types such as native mobile applications. When these credentials are long-lived and can be “lifted and shifted” from the machine where they were created to be usable by a bad actor from another machine, their tradable value is significant. Emerging technologies such as Device Bound Session Credentials (DBSC) for browsers and Demonstrating Proof of Possession (DPoP) for OAuth applications seek to reduce the threat of session hijacking. This article describes how these technologies address the problem of session hijacking and how they complement strong phishing resistant authentication in online ecosystems.

Audience

This white paper is for chief information security officers (CISOs) and technical staff whose responsibility it is to protect the security and life cycle of online identity and access management from online fraud.

Table of Contents

1 Introduction

Authentication and authorization are integral parts of an identity lifecycle, especially for online credential ecosystems. The growing threat of online identity fraud with costly security incidents and breaches has enterprises looking for ways to protect and secure their workforces from account takeover through different attack vectors such as phishing, credential stuffing, and session hijacking. For authentication, FIDO authentication with passkeys provides users with “Safer, more secure, and faster online experiences”[1], and an increase in the adoption of passkeys has contributed to a reduction of the success of attack vectors of credential phishing, credential stuffing, and session hijacking accomplished via man-in-the-middle (MITM) phishing attacks. However, what happens after the authentication ceremony?

After authentication, browsers and application clients are typically issued other credentials. Enterprise applications generally fall into two primary categories: those that are web browser based and use session cookies for state management and those that are thick client applications using OAuth access tokens (this includes some browser-based single page applications and most native mobile applications). Both types of credentials (session cookies and access tokens) are considered, in their basic use, as “bearer” tokens. If you have the token (the session cookie or the access token), then you can continue to transact for the lifetime of that token as the user who authenticated and owned it.

This whitepaper explores adjacent technologies that address the “lift and shift” attack vector for bearer tokens and how these technologies complement FIDO-based authentication mechanisms. In particular, this paper focuses on the proposed web standard **Device Bound Session Credentials (DBSC)**[2] for protecting browser session cookies and ***OAuth 2.0* Demonstrating Proof of Possession (DPoP)**[3] for protecting OAuth grants.

2 Terminology

session hijacking: An exploitation of the web session control mechanism that is normally managed for a session cookie[4].

credential stuffing: An automated injection of stolen username and password pairs (credentials) into website login forms to fraudulently gain access to user accounts[5].

---

[1] Passkeys - https://fidoalliance.org/passkeys/
[2] Device Bound Session Credentials - https://github.com/w3c/webappsec-dbsc
[3] OAuth 2.0 Demonstrating Proof of Possession (DPoP) - RFC9449: https://datatracker.ietf.org/doc/html/rfc9449
[4] Session hijacking attack https://owasp.org/www-community/attacks/Session_hijacking_attack
[5] Credential stuffing https://owasp.org/www-community/attacks/Credential_stuffing

access token: A credential used by a client-side application to invoke API calls on behalf of the user.

session cookie: A credential managed by browsers to maintain session state between a browser and a website.

bearer token: A token (in the context of this whitepaper may refer to either an access token or a session cookie) so called because whoever holds the token can use it to access resources. A bearer token on its own can be “lifted and shifted” for use on another computing device.

sender-constrained token: A token protected by a mechanism designed to minimize the risk that anything other than the client which established the token during an authentication process could use that token in subsequent requests for server-side resources.

Device Bound Session Credential (DBSC): A proposal for a W3C web standard defining a protocol and browser behavior to establish and maintain sender-constrained cookies. The mechanism uses proof of possession of an asymmetric cryptographic key to help mitigate session cookie hijacking.

OAuth 2.0 Demonstrating Proof of Procession (DPoP): A mechanism for implementing sender-constrained access tokens that requires clients to demonstrate possession of an asymmetric cryptographic key when using the token.

3 Adjacent/complementary technologies for a secure ecosystem

While FIDO authentication technology can effectively eliminate phishing and credential stuffing attacks that occur during the login process, the addition of solutions to mitigate threats associated with bearer token theft is equally important. Bad actors whose attacks are thwarted during the login process will go after the next weakest link in the chain and try to steal post-authentication bearer tokens. This section explores two of these technologies for protecting bearer tokens: Device Bound Session Credentials (DBSC) protect browser-based session cookies and Demonstrating Proof of Possession (DPoP) protects OAuth grants. Alternative approaches to protect bearer tokens are also discussed.

Because no single piece of technology can protect against all threats, a combination of multiple techniques is required for adequate protection.

Table 1: Combination of technologies for increased security

Technologies	Authentication threats		Post-authentication threats
	Remote Phishing	Credential Stuffing	Token Theft
Passkeys	✅	✅	❌
DBSC/DPoP	❌	❌	✅
Passkeys + DBSC/DPoP	✅	✅	✅

3.1 Browser session cookie security

Before discussing Device Bound Session Credentials (DBSC), you will need to understand the problem being addressed regarding browser session cookies. Session hijacking via cookie theft allows an attacker, who possesses stolen cookies, to bypass end-user authentication, including any strong or multi-factor authentication (MFA). This is particularly problematic when browsers create long-lived session cookies (which are a type of bearer token), since these cookies can be traded as alternatives to a user's primary authentication credentials and then used from the attacker's machine. This can lead to unauthorized access to sensitive data, financial loss, and damage to an organization's reputation.

Attackers perform cookie theft through various methods such as man-in-the-middle phishing of a user's existing MFA login process (when phishing-resistant authentication such as FIDO is not used), client-side malware, and occasionally through vulnerabilities in server-side infrastructure or software. Regardless of how cookie theft is perpetrated, when successful, these attacks are not only dangerous, but also hard to isolate and detect. Complementary technologies, such as Device Bound Session Credentials (DBSC), minimize the risks associated with browser cookie theft by making stolen cookies impractical to use from any machine other than the machine to which they were issued during authentication.

3.2 Device Bound Sessional Credentials - DBSC

DBSC refers to a proposed web standard currently in development within the Web Application Security working group of the W3C[2]. The goal of DBSC is to combat and disrupt the stolen web session cookies market. This is achieved by defining an HTTP messaging protocol and required browser and server behaviors to result in binding the use of application session cookies to the user's computing device. DBSC uses an asymmetric key pair and in browser implementations the private keys should be unextractable by an attacker - for example stored within a Trusted Platform Module (TPM), secure element, or similar hardware-based cryptographic module.

At a high level, the API in conjunction with the user's browser and secure key storage capabilities, allows for the following:

- The server communicates to the browser a request to establish a new DBSC session. This includes a server-provided challenge.
- The browser generates an asymmetric key pair, then sends the public key along with the signed challenge to the server. This process is referred to as DBSC registration. Browser implementations of DBSC should use operating system APIs that facilitate secure, hardware-bound storage and use of the private key.
- The server binds the public key to the browser session by issuing a short-lived, refreshable `auth_cookie` which is then required to be transmitted in subsequent browser requests to the web server.

As the `auth_cookie` regularly expires, a mechanism is required for the browser to refresh the `auth_cookie` asynchronously to primary application web traffic. The refresh process requires signing a new server-issued challenge with the same private key created during DBSC registration, thereby re-proving (regularly) that the client browser is still in possession of the same private key.

Limiting the lifetime of the `auth_cookie` to short periods of time (for example, a few minutes) disrupts the market for trading long-lived session cookies. An attacker can only use stolen session cookies (including the `auth_cookie`) for a brief period, and cannot perform a refresh operation, since the private key required to perform a refresh operation is not extractable from the client machine.

DBSC may be introduced into existing deployments with minimal changes to the application. This is important as DBSC could easily be incorporated as a web plugin module in existing server-side technology (for example, Apache module, Servlet Filter, or reverse proxy functionality). This permits enterprises to roll out deployment of DBSC in phases without a complete overhaul of all current infrastructure and companies can prioritize certain critical endpoints or resources first.

DBSC server-side implementations can also be written in a manner that permits semantics, for example: "If the browser supports DBSC, use it, otherwise fallback to regular session characteristics." This allows users to gain the security advantages of DBSC when they use a browser that supports it without having to require all users to upgrade their browsers first.

Refer to the Device Bound Session Credentials explainer for more details on the DBSC protocol and standard, including a proposal for enterprise-specific extensions that adds attestation to DBSC keypairs.

3.2.1 What makes DBSC a technology complementary to FIDO?

The DBSC draft standard permits the login process to be closely integrated with the DBSC API. While FIDO is a mechanism that makes authentication safer and phishing resistant, DBSC is a mechanism that makes the bearer credential (session cookie) safer post-authentication. They complement each other by reducing the risk of account takeover and abuse, making the entire lifecycle of application sessions safer.

3.2.2 Alternative solutions

DBSC is not the first standard to propose binding session cookies to a client device. Token Binding is an alternative that combines **IETF RFCs 8471**, **8472**, and **8473**. Token Binding over HTTP is implemented via a Transport Layer Security (TLS) extension and uses cryptographic certificates to bind tokens to a TLS session. Token Binding has had limited browser adoption and is complex to implement as it requires changes at the application layer and in TLS security stacks. The Token Binding over HTTP standard has not been widely adopted and only one major browser currently offers support.

3.2.3 Advice

The DBSC standard relies on local device security and operating system APIs for storage and use of the private key that is bound to the browser's session. While these private keys cannot be exported to another device, the key is available on the local system and may be exercisable by malware residing on the user's device. Similarly, in-browser malware still has complete visibility into both regular session cookies and short-lived `auth_cookies`. DBSC is not a replacement for client-side malware protection, and the threat model for DBSC does not provide protections from persistent client-side malware. Ultimately, the user must trust the browser.

As browsers start to support DBSC over time, it will be important for servers to be able to work with a mix of browsers that do and do not include support for this technology. Some enterprises may dictate that corporate issued machines include browsers known to support DBSC, but many will not. It will be necessary for server-side implementations to take this into consideration, using DBSC when the browser responds to registration requests, and tolerating unbound session cookies when the browser does not. When building or choosing a commercial solution, ensure you consider this scenario, and include the ability to implement access control policies that strictly require DBSC in highly controlled or regulated environments or for specific applications.

At the time of writing, DBSC is in early evolution. It remains to be seen whether or not it will be widely adopted by browser vendors. The hope is that incubating and developing this standard via the W3C will result in wider adoption than previous proposals, similar to the way that the WebAuthn API has been adopted to bring passkey authentication to all major browser implementations.

4 OAuth grants

The previous section introduced DBSC as a means to protect against session cookie theft in web browsers. Thick application clients, including mobile applications and single-page web applications, typically use stateless API calls leveraging OAuth grants instead of session cookies. An OAuth grant may be established in several ways, with the recommended pattern for thick clients[6] being to initially use the system browser to authenticate a user, and grant access for an application to act on their behalf. Conceptually this is remarkably similar to browser-based sessions, including the ability and recommendation, to use FIDO authentication for end-user authentication when possible. At the conclusion of the browser-based authentication portion of this flow, control is returned to the thick client application or single-page web application where tokens are established for use in programmatic API calls.

The challenge that occurs from this point forward is almost identical to that described for browsers - the OAuth tokens are bearer tokens that if exposed to a bad actor can be used to call application APIs from a remote machine instead of from the legitimate application.

This section describes the use of DPoP, a technology for protecting the “lift and shift” of credentials used in OAuth-protected API calls which, just like DBSC, makes use of an asymmetric key pair and ongoing proof of possession of the private key.

4.1 Demonstrate Proof of Possession (DPoP)

OAuth 2.0 Demonstrating Proof of Possession (DPoP) is an extension of the existing OAuth 2.0 standard for implementing device bound (or sender-constrained) OAuth access and refresh tokens. It is an application-level mechanism that allows for the tokens associated with an OAuth grant (that is, refresh tokens and access tokens) to bind with the requested client using a public and private key pair. This requires the client to prove ownership of its private key to the authorization server when performing access token refresh operations and to resource servers when using access tokens to call APIs.

[6] OAuth 2.0 for Native Apps https://datatracker.ietf.org/doc/html/rfc8252

High assurance OpenID specifications, such as Financial-grade API (FAPI 2.0), mandate the use of sender-constrained tokens and DPoP is the recommended method for implementing this requirement when Mutual TLS (mTLS) is not available.

At a high level, DPoP requires that:

- The client generates a per-grant public/private key pair to be used for constructing DPoP proofs. Best practice implementations should use operating system APIs to ensure the private key is non-extractable.
- On initial grant establishment (for example, exchanging an OAuth authorization code for the grant's first access token and refresh token), a DPoP proof (a JWT signed by the client's private key that contains, among other things, a copy of the public key) is used to bind a public key to the grant.
- Requests to a resource server using an access token obtained in this manner must also include a DPoP proof header, continuously proving possession of the private key used during grant establishment. This is done for every API request. Resource servers are required to check if an access token is sender-constrained, confirm the public key, and validate the DPoP proof header on each API call.
- For public clients, subsequent `refresh_token` flows to the authorization server's token endpoint must also contain a DPoP proof signed with the same key used during initial grant establishment. This is particularly important as the refresh tokens are often long-lived and are also a type of bearer token (that is, if you have it you can use it). The authorization server must enforce the use of a DPoP proof for these refresh token flows and ensure signature validation occurs via the same public key registered during initial grant establishment.

Unlike a plain bearer access token which can be used by any holder, DPoP based access tokens are bound to the client that initially established the OAuth grant, since only that client can sign DPoP proofs with the private key. This approach minimizes the risks associated with malicious actors trading leaked access tokens.

Refer to DPoP RFC 9449 - OAuth 2.0 Demonstrating Proof of Possession (DPoP) for more information.

4.2 What makes DPoP a complementary technology to FIDO?

FIDO can be leveraged for phishing resistant end-user authentication during establishment of an OAuth grant. Refresh and access tokens obtained by a client following this authentication should be safeguarded against "lift and shift" attacks just like session cookies in browser-based apps. DPoP is a recommended solution for protecting these OAuth tokens from unauthorized post-authentication use. Together, FIDO for end user authentication and DPoP for binding OAuth tokens to a client device complement each other to improve the overall security posture for identities used in thick client applications.

4.2.1 DPoP alternative solutions?

RFC8705 - OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens describes a mechanism that offers a transporter layer solution to bind access tokens to a client certificate. While it has been approved for use in FAPI 2.0 for open banking solutions, it is not particularly suitable for public clients such as native mobile applications.

RFC9421 - HTTP Message Signatures defines an application-level mechanism for signing portions of an HTTP message. Key establishment and sharing between the client and verifier are not defined by this specification, although this could be performed in a *trust on first user* manner during initial grant establishment in a similar manner to DPoP. There is no known public specification that maps the use of HTTP message signatures to the use case of sender-constrained bearer tokens in an OAuth client application. In the absence of such a public specification, widespread adoption for this use case is unlikely.

4.2.2 Advice

Sender-constrained tokens are a good idea, and, in some deployments, they are a regulatory requirement. For example, use of the FAPI profiles of OAuth is now mandated by many sovereign open banking initiatives. DPoP is a relatively simple way to achieve this requirement and is flexible enough to cover a wide range of application client types. That said, care must still be taken to adhere to the security considerations of DPoP. Pay close attention to section 11 of RFC9449, as well as apply other application security strategies for native or browser based single page applications as your scenario dictates. Remember that DPoP is focused solely on addressing the threats associated with token exfiltration, which include trading and use by malicious actors. It should be considered part of a defense-in-depth strategy for OAuth applications.

5 Conclusion

The intent of this paper is to inspire thinking around how different web security standards fit together and how those standards relate to the use of FIDO authentication for users. There are so many standards and standards bodies that it is often hard to understand which compete in the same space and which augment one another to form part of a comprehensive defense-in-depth strategy for identity fraud protection in online applications.

This paper tackled a specific, prevalent application security problem - the malicious trading and use of stolen cookies and access tokens. This paper also showed how technologies such as DBSC and DPoP mitigate the threats associated with token theft and how these technologies are complementary to FIDO authentication. Paired with FIDO, DBSC and DPoP provide greater overall identity fraud protection for your applications.

Document history

Change	Description	Date
Initial publication	White paper first published.	May 2025