A leap forward in token security: Okta adds support for DPoP

By About the Author

Have you ever wondered how secure your tokens are when they are transmitted from a client to a server? As our digital footprints expand, the need for more stringent security measures in web applications and APIs has never been greater. In the wake of a recent GitHub security breachopens in a new tab affecting dozens of organizations, the question of token securityopens in a new tab has once again been thrust into the spotlight.

Recently, a group of hackers used stolen OAuth access tokensopens in a new tab to gain unauthorized access to numerous organizations' internal systems, causing considerable disruption and raising serious concerns over how we protect our digital identities. This incident serves as a stark reminder of the importance of implementing robust security measures, especially when it comes to OAuth tokens.

Today, we are excited to announce support for the OAuth 2.0 Demonstrating Proof-of-possession at the Application Layer (DPoP)opens in a new tab, a significant approach for bolstering security and mitigating risks associated with token-based authentication. In this blog post, we will delve deep into the world of OAuth 2.0 token binding using cryptographic keys. We will explore what it means, how it works, and why it is an important step toward ensuring a more secure token transmission process.

History of API security

In the early days of API development, API keys were the cornerstone of authentication. These static strings were passed along with API calls, offering a simple yet effective way to identify and authenticate the client. But, as the digital landscape evolved, so did the security demands. Despite the convenience of API keys, they come with notable downsides. They never expire and lack authorization information, which means they can be used to perform any operation. This open-ended nature of API keys poses considerable security risks, especially if they fall into the wrong hands.

Enter the world of OAuth 2.0, a protocol designed to overcome these limitations. OAuth access tokens are short-lived strings that can be used to make API calls. They are ephemeral, with shorter expiration times than API keys, and they carry information about the user and the application to which the token was minted. Moreover, access to APIs can be restricted using OAuth scopes and permissions. Despite these advancements, OAuth tokens, like API keys, are still bearer tokens. This means anyone who gains possession of these tokens can use them just as any other party could — to make API requests and extract information. Given the far-reaching implications of a token breach, it’s clear that we need a way to ensure that even if tokens are intercepted, they can't be misused.

That's where the new OAuth 2.0 standard comes in. In a significant stride towards enhancing token security, Okta has introduced support for DPoP with our OAuth applications and authorization servers.

Proof of possession

Proof-of-possession (PoP) in OAuth tokens refers to a security mechanism that ensures the client sending a request to the resource server is in possession of a specific cryptographic key, thus proving its identity. This concept is particularly important in the context of OAuth 2.0, as it aims to improve the security of bearer tokens, which are the common choice. There are two ways to implement proof of possession for OAuth tokens, mTLS-based certificate-bound access tokens (RFC 8705)opens in a new tab and DPoPopens in a new tab.

Mutual Transport Layer Securityopens in a new tab (mTLS) PoP is a security mechanism that ensures both the client and server involved in a communication authenticate each other using digital certificates, providing an extra layer of security compared to regular TLS. In regular TLS, only the server presents a certificate for authentication. However, in mTLS, both client and server present their certificates, ensuring that both parties are who they claim to be. mTLS can be used with certificate-bound access tokens, which bind access tokens to specific cryptographic keys and require the sender to prove possession of those keys. But the mTLS-based proof of possession comes with its own disadvantages.

Complexity

Implementing mTLS requires special domain knowledge and additional labor, which could lead to a steeper learning curve for developers, especially for those who are not well-versed in mTLS.

Compatibility issues

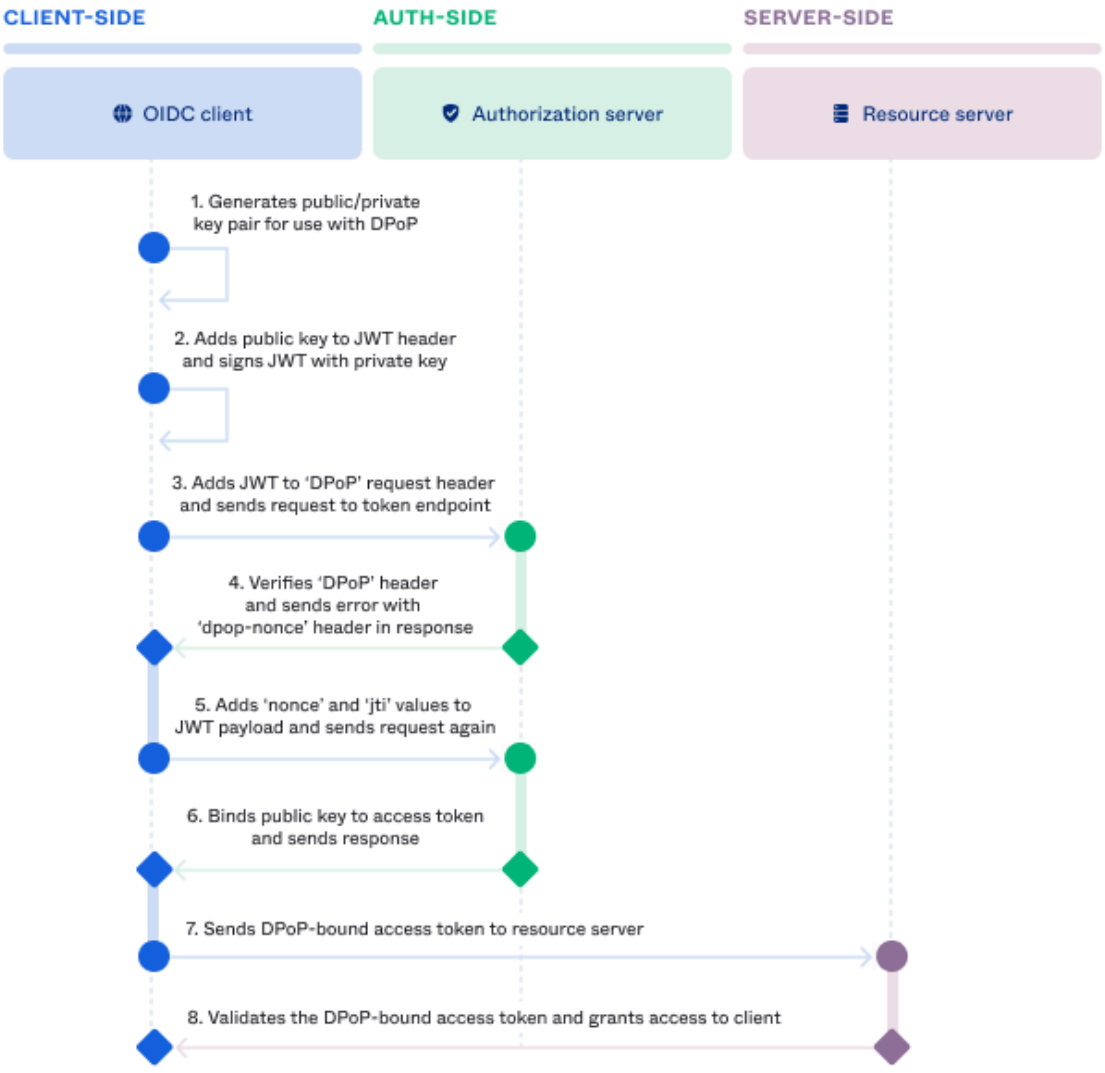
Although mTLS offers strong security, it may not be compatible with all tech stacks or platforms. For instance, mTLS may not be suitable for certain client types, such as single-page applications (SPAs) and native applications (Mobile Apps), as it's not easily implementable in these scenarios. This is primarily due to client-side storage limitations and browser compatibility issues.

Performance Impact

While mTLS does not incur an extra roundtrip for client authentication, it does add some overhead in terms of data during the initial handshake. This additional data can potentially affect performance, especially when dealing with large-scale systems or high volumes of requests.

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

Just like mTLS, DPoP provides better security compared to traditional bearer tokens by binding access and refresh tokens to a private key that belongs to the client. This binding makes the DPoP access token sender-constrained, meaning that its replay, if leaked or stolen, can be detected and prevented, unlike the common bearer token.



The client first generates a public/private key pair for use with DPoP. Then the client adds the public key in the header of the JWT and signs the JWT with the private key. The client adds the JWT to the DPoP request header and sends the request to the token endpoint for an access token. After verifying the DPoP header, the authorization server binds the public key to the access token and sends the response.

When the client accesses a protected API using a DPoP token, it generates a new DPoP proof signed by the same private key used to obtain the token. This proof must be included as part of the DPoP header, which provides an additional layer of security compared to bearer tokens. DPoP-bound tokens mitigate the misuse of tokens by unauthorized parties, including bad actors and resource APIs with access to the token that re-use it to access other endpoints.

Benefits of using DPoP over mTLS for token protection

Since DPoP operates at the application layer, leveraging asymmetric cryptography and lightweight JSON Web Tokens (JWTs), it is much more accessible to developers, as it does not require additional infrastructure and is

transparent to end users. DPoP also works well with public OAuth 2.0 clients, such as single-page applications (SPAs) and mobile applications.

How can you get started?

We are thrilled to announce out-of-the-box support for OAuth DPoP token protection for all Okta customers. To get started, follow the DPoP developer guideopens in a new tab.
