**[Client Secret](https://www.rfc-editor.org/rfc/rfc6749" \l "section-2.3.1) (RFC 6749 Section 2.3.1)**

**HTTP Basic Authentication:**

Clients that have a "client password" can use HTTP Basic Authentication to authenticate themselves. This means they will send a username (the client identifier) and password (the client password) to the server for verification.

**Encoding:**

Both the client identifier (username) and client password are encoded using a specific format, which is called "application/x-www-form-urlencoded." This is just a way of transforming the information into a format that can be safely sent over the internet.

**Authorization Server Requirements**

The server receiving the authentication request must support this HTTP Basic Authentication scheme to verify the client’s identity.

**Example of Basic Authentication**:

A sample "Authorization" header might look like this:

Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3 This is just an example of the encoded client identifier and password.

**Alternative Authentication Method:**

Alternatively, instead of using HTTP Basic Authentication, the client can send the client credentials (client ID and client secret) directly in the request body.

client\_id: This is the unique identifier for the client.

client\_secret: This is the secret password for the client. If it's empty, the client can leave it out.

**Recommendation on Usage:**

Sending credentials in the request body (instead of HTTP Basic Authentication) is NOT recommended unless the client can't use the Basic Authentication method. It’s less secure because it could potentially expose the credentials in ways Basic Authentication does not.

**Security Requirements**:

The authorization server must enforce TLS (Transport Layer Security), a security protocol, when receiving these authentication requests to protect the credentials.

The server also needs to be protected against brute-force attacks, where attackers try multiple passwords to guess the correct one.

**RFC 8705: Mutual TLS Client Authentication and Certificate-Bound Access Tokens (MTLS)**

### 1. ****What is MTLS?****

* **MTLS** stands for **Mutual TLS** (Transport Layer Security).
* It is a form of **client authentication**, where both the client **and** the server authenticate each other.
  + In typical SSL/TLS, only the server proves its identity to the client (via a server certificate).
  + With MTLS, both the **client** and the **server** authenticate each other by using certificates. The client has to prove its identity by presenting a **client certificate**, and the server verifies it.

### 2. ****How does MTLS work with OAuth 2.0?****

* OAuth 2.0 is a framework for granting access to resources without sharing passwords. It uses **Bearer Tokens** to give clients access to resources.
* MTLS is an **extension of OAuth 2.0**, meaning it works together with OAuth 2.0 to improve security by adding an additional authentication layer.
* Instead of just sending a **Bearer Token** to authenticate, MTLS requires the **client application** to authenticate itself by presenting a valid **client certificate**. This helps to ensure that the application is who it says it is, not just anyone with a valid token.

### 3. ****Binding Tokens to Certificates:****

* MTLS also **binds the access token** to the client’s certificate. This means the token is **tied** to a specific client (the one holding the certificate).
* Even if someone else gets a hold of the token, they won't be able to use it unless they also have the **correct client certificate**. This adds another layer of security, because the access token alone is not enough to access the resource.

### 4. ****Security Enhancement:****

* One of the primary motivations for MTLS is to improve the security of **Bearer Tokens**.
  + Typically, Bearer Tokens can be misused if they are intercepted, because anyone who possesses the token can use it to access resources.
  + MTLS makes this much harder because, in addition to the token, the application must also **prove** its identity using its **client certificate**. This means that the application is **authenticated** and the server can trust that the request is from the right source.

### In summary:

* MTLS is a security technique that requires both the client and the server to authenticate each other using certificates, and it is an **extension of OAuth 2.0**.
* It **binds** access tokens to the client certificate, making the token more secure and ensuring that only the legitimate client can use it.
* It improves the security of OAuth 2.0 by preventing misuse of tokens if they are intercepted or stolen.

## Private Key JWT

The **Private Key JWT** method is another way for clients to authenticate themselves when interacting with authorization servers in protocols like **OAuth 2.0**.

### 1. ****What is Private Key JWT?****

* **JWT** stands for **JSON Web Token**. It's a way of securely transmitting information between parties in the form of a compact, URL-safe token.
* **Private Key JWT** means that the client **creates** and **signs** the JWT using its own **private key**. A private key is part of a cryptographic key pair: the private key is kept secret by the client, and a matching public key can be shared with others for verification.

### 2. ****How Does Private Key JWT Work?****

* The client first **creates** a JWT that contains some details (like its identity and any claims or data needed).
* Then, it **signs** this JWT with its **private key**. The **signature** proves that the JWT was indeed created by the client (because only the client has the private key) and hasn't been tampered with.
* The signed JWT is then sent to the authorization server for **authentication**.
* The server can verify the JWT by checking the **signature** using the **client's public key** (which the server should already have or be able to retrieve).

### 3. ****Where is the JWT Sent?****

* Even though this authentication method is called **Private Key JWT**, the JWT is not directly sent in the request body as a typical JWT. Instead, it is sent in a **parameter called client\_assertion**.
* This means that, in the context of OAuth, when the client makes a request to the authorization server, it sends the **JWT** as part of the request, specifically in the client\_assertion field.

### 4. ****What’s the Relationship with RFCs?****

* The method of **Private Key JWT** is described in **RFC 7521** (Assertion Framework for OAuth) and **RFC 7523** (JWT Profile for Client Authentication).
* **OpenID Connect** and **FAPI 2.0** (Financial-grade API security) also reference these RFCs as part of their security guidelines. These protocols and security profiles help to define best practices for using JWTs securely in client-server authentication.

### 5. ****Security Consideration:****

* By using a **private key** to sign the JWT, this method ensures that the **client** is securely proving its identity without transmitting sensitive information like a password.
* Only the **client** holding the private key can sign the JWT correctly, making it very difficult for an attacker to impersonate the client, even if they intercept the JWT.

### Summary:

* **Private Key JWT** is a way for a client to authenticate with a server by creating a **JWT** (JSON Web Token) and **signing** it with its **private key**.
* This method is used in OAuth 2.0 and related protocols to enhance security, as it ensures the client is who it claims to be without sending sensitive information like passwords.
* The signed JWT is sent in the **client\_assertion** parameter in requests.
* It is described in **RFC 7521** and **RFC 7523**, and is used in security profiles like **OpenID Connect** and **FAPI 2.0**.