In the context of client credential flow with certificates in Microsoft Entra (formerly Azure Active Directory), it’s essential to understand the roles of the various keys involved, especially regarding the JWT (JSON Web Token) and the signing and validation processes. Here's a detailed breakdown:

**Key Concepts and Relationships**

1**. Private Key and Public Key Pair:**

- Private Key: The private key is used by the client application (the one requesting the JWT) to sign the JWT token. This key must be kept secure and should never be shared or exposed.

- Public Key: The corresponding public key is shared (or accessible) and is used by the resource server (or any other service that needs to validate the JWT) to verify the signature of the token. This verification process ensures that the token was indeed issued by the entity holding the private key.

**2. Certificates in MS Entra:**

- When you register an application in Microsoft Entra, you can upload a certificate, which contains the public key. This public key is stored in the application’s registration and is used by Azure AD to validate the authenticity of tokens signed by the private key of the certificate.

- The public key in the certificate is part of the signing keys that Azure AD uses for token validation.

**3. OpenID Connect (OIDC) Well-Known Configuration Endpoint:**

- This endpoint provides metadata about the Azure AD instance, including information about the token issuer, authorization endpoints, and, crucially, the signing keys.

- The signing keys listed in the **.well-known** metadata are the public keys that Azure AD uses to sign tokens issued to clients. These keys are typically rotated every 24 hours, as you noted.

**Key Usages in the Client Credential Flow with Certificates**

**1. Requesting a JWT:**

- The client application generates a JWT using its private key. This JWT is constructed to contain claims about the application, including the intended audience, issuer, expiration time, and more.

**2. Signing the JWT:**

- The JWT is signed with the private key. This means that any service (like Azure AD) that has access to the corresponding public key can validate that the token was indeed created by the application that possesses the private key.

**3. Token Issuance:**

- The client application sends the signed JWT to Azure AD when requesting an access token. Azure AD verifies the signature using the public key associated with the application’s certificate.

**4. Token Validation:**

- When another service receives the JWT (for instance, an API), it checks the token’s signature using the public key from the Azure AD **.well-known** configuration endpoint. This confirms that the token is valid and was issued by Azure AD.

- The service can also check the token's claims (such as audience and expiration) to ensure that it is valid for the intended recipient and has not expired.

**5. Key Rotation:**

- If the signing keys in Azure AD are rotated (every 24 hours as you mentioned), the public key available at the .**well-known** endpoint will change. This means that any service validating JWTs must fetch the latest keys to ensure successful validation.

- In practice, applications that validate tokens typically implement a caching mechanism to optimize the retrieval of these keys, updating their cache only when they detect that the keys have changed.

**Summary of Relationships:**

- **Private Key** → Used to sign the JWT.

- **Public Key (from the certificate)** → Stored in MS Entra, used to verify the JWT's signature.

- **Signing Keys from the OIDC Metadata** → Public keys used by Azure AD to validate JWTs issued to clients; these are updated every 24 hours and should be refreshed in any validation logic in the consuming applications.

**Final Notes**

The security of the client credential flow heavily relies on the proper management of these keys and certificates. It’s critical to ensure that the private key is secured and that any public keys used for validation are up-to-date and correctly configured in the services consuming the JWT.

**1. Public Key from the Certificate (Your App's Certificate):**

- When you upload a certificate as part of your app's registration in Azure AD, this certificate contains a public key that corresponds to the private key your application uses to sign JWTs when making client assertions in the client credentials flow.

- This public key is used by Azure AD to verify that the JWT (client assertion) sent by your application is valid and was indeed signed by the private key associated with the registered certificate.

**2. Signing Keys from the OIDC Metadata Endpoint:**

- The signing keys exposed in the `.well-known` OIDC metadata endpoint are the public keys used by Azure AD to sign tokens it issues to clients and APIs (like access tokens, ID tokens, etc.).

- When Azure AD issues a token (for example, an access token or ID token), it signs that token with its private key. The corresponding public key is published in the OIDC metadata endpoint so that resource servers (APIs) can verify the authenticity of tokens they receive from Azure AD.

**Key Points:**

- Your app's certificate's public key (uploaded in the app registration) is used only by Azure AD to validate tokens (JWT client assertions) that your app signs with its private key when making requests to obtain access tokens.

- The OIDC metadata signing keys are used by other applications or services (such as APIs) to verify tokens that Azure AD issues. These tokens are signed by Azure AD using its private key, and the corresponding public key is available at the OIDC metadata endpoint.

**Summary of Separation:**

- Your certificate's public key: Used by Azure AD to validate your app's client assertions.

- OIDC metadata signing keys: Used by resource servers (APIs) to validate tokens issued by Azure AD to your app or other clients.

They are two different sets of keys used in distinct parts of the authentication process, and they do not interact with each other.

The explanation I provided is based on the standard practices of the OAuth 2.0 client credentials flow, the use of JWTs, and the way Azure AD (now Microsoft Entra) handles certificates and token signing. These concepts are derived from public documentation and protocols for OAuth 2.0 and OpenID Connect (OIDC), which Microsoft Entra adheres to. Specifically:

**1. OAuth 2.0 Client Credentials Flow:**

- [OAuth 2.0 Client Credentials Grant Flow](https://datatracker.ietf.org/doc/html/rfc6749#section-4.4) (RFC 6749)

- This flow explains how a client authenticates using a JWT signed by its private key, which Azure AD verifies using the corresponding public key.

**2. JWT and Asymmetric Key Validation:**

- [JWT Profile for OAuth 2.0](https://datatracker.ietf.org/doc/html/rfc7523) (RFC 7523)

- This explains how a JWT is signed with the private key and validated using the public key, forming the basis of client assertions and token validation in the client credentials flow.

**3. Azure AD Certificate-Based Authentication:**

- [Microsoft Documentation: Use certificates in Azure AD applications](https://learn.microsoft.com/en-us/azure/active-directory/develop/active-directory-certificate-credentials)

- Describes how certificates are used in app registrations, where the public key is uploaded, and how Azure AD uses it to verify client assertions.

4. **Azure AD Token Signing and Key Rotation:**

- [Microsoft Documentation: OpenID Connect Metadata](https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-protocols-oidc)

- Details how Azure AD signs the tokens it issues, and how public keys (used for validating Azure AD-issued tokens) are made available through the **.well-known** OIDC metadata endpoint.

These references together explain the separation of concerns between your app's private/public key pair (used for client assertions) and Azure AD's signing keys (used for validating tokens issued by Azure AD). While I haven't cited a single document that directly compares these key sets, these sources outline the relevant roles of keys in both processes.