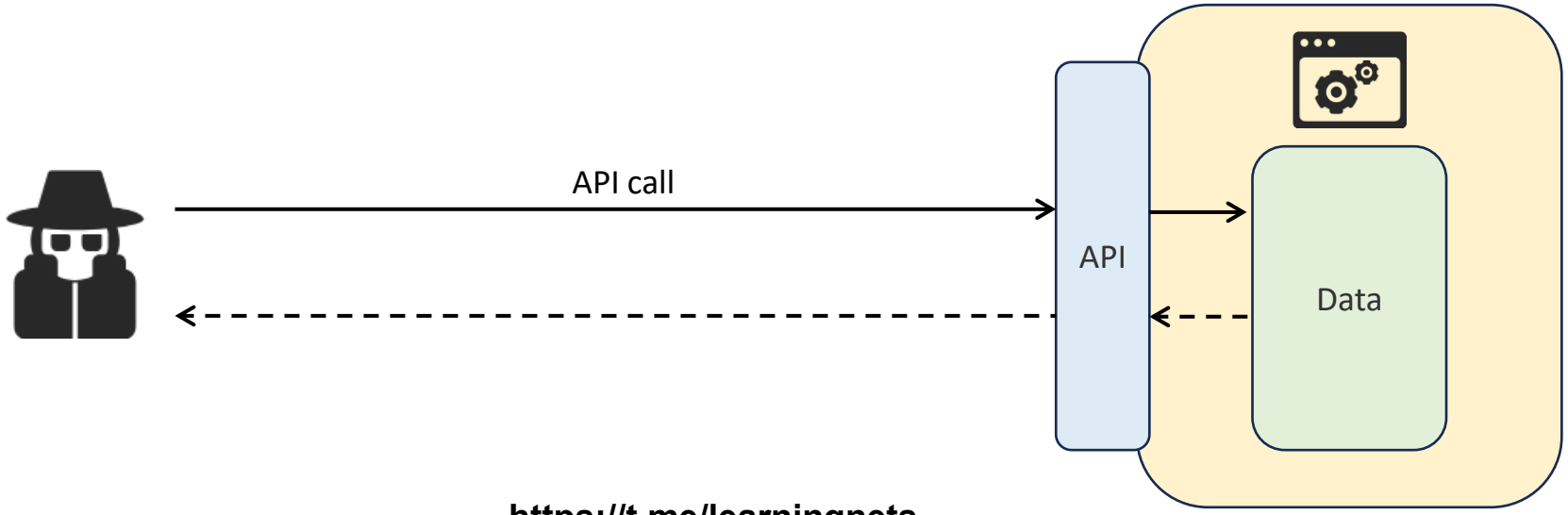
REST API Authentication

- **Authentication** is the process of validating the identity of a user or system to ensure legitimate access to resources.
  - In the case of an API, the resource is the application and its data.

- Without authentication, unauthorized users can send API requests, potentially accessing sensitive data or modifying the application.

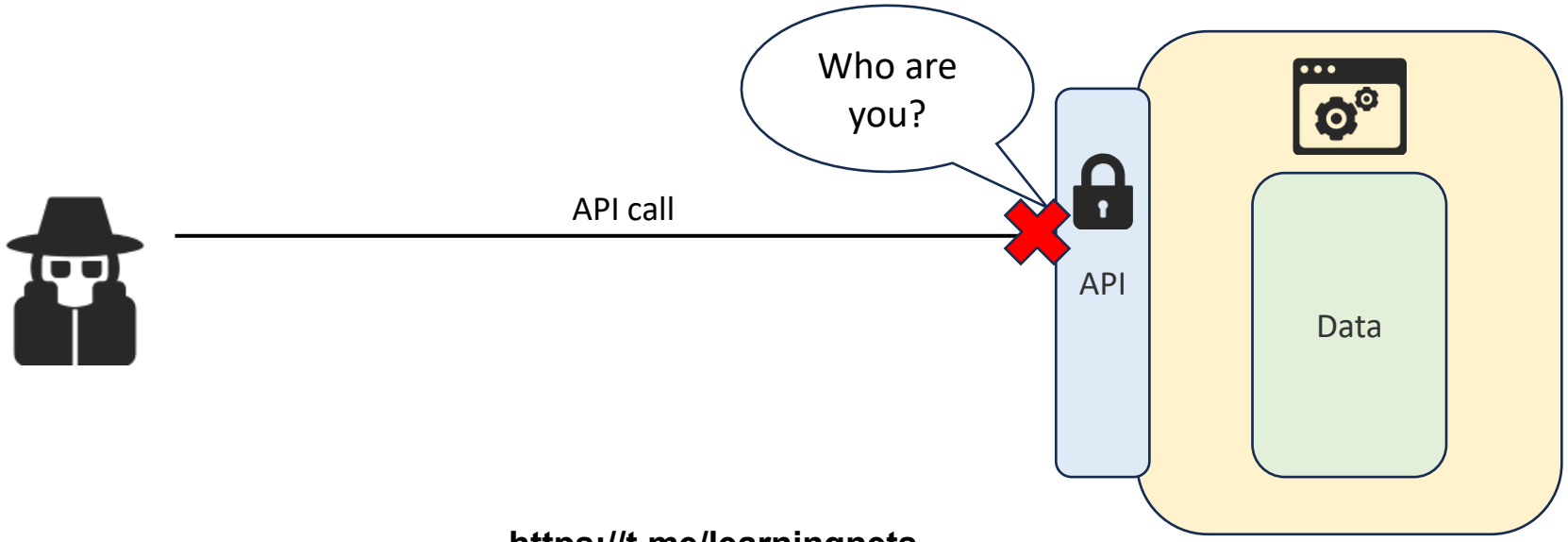


REST API Authentication

- **Authentication** is the process of validating the identity of a user or system to ensure legitimate access to resources.
  - In the case of an API, the resource is the application and its data.

- Without authentication, unauthorized users can send API requests, potentially accessing sensitive data or modifying the application.
  - Implementing a reliable authentication method is essential for protecting applications and data.

- Many APIs track usage for analytics and billing purposes.
  - e.g., charging customers according to how much they use the API.



Types of REST API authentication

- REST APIs use various types of authentication to verify client identity and secure access to resources.
  - These are also called **methods** or **schemes**.

- We will cover four:
  - **Basic authentication:**
    - Sends a username and password in every request, encoded in Base64.
  - **Bearer authentication:**
    - Uses a token (bearer token) as an HTTP header in each the request to verify the client's identity.
  - **API key authentication:**
    - Requires a unique key, typically included as an HTTP header, to authenticate API requests.
  - **OAuth2.0:**
    - A secure framework that grants access via access tokens, commonly used for delegated access and third-party authentication.

Basic authentication

- **Basic authentication** includes a username and password in the HTTP headers of each API request for authentication.

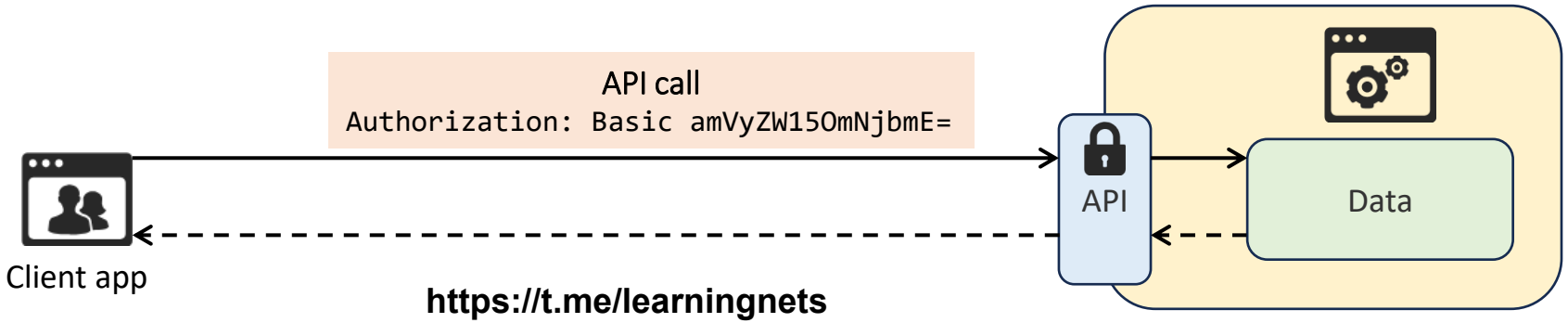
Basic authentication

- **Basic authentication** includes a username and password in the HTTP headers of each API request for authentication.

- These credentials are **encoded** in Base64 format but not **encrypted**.
  - **Base64** is an encoding scheme, which is simply a way of representing data.
  - Unlike encryption, it is not secure and can easily be decoded.
  - Always use HTTPS (TLS) for security.

- The username/password are sent in the format *`username:password`*, encoded in Base64.
  - For example, `jeremy:ccna` would be sent as `amVyZW15OmNjbmE=`
  - You can encode/decode Base64 at https://www.base64decode.org/

- **Advantages:**
  - Simple and easy to implement.
- **Disadvantages:**
  - Since credentials are sent in every request, attackers could steal them if the connection is not properly secured.
  - Even if using HTTPS for encryption, relying solely on a username/password combination isn't particularly secure.



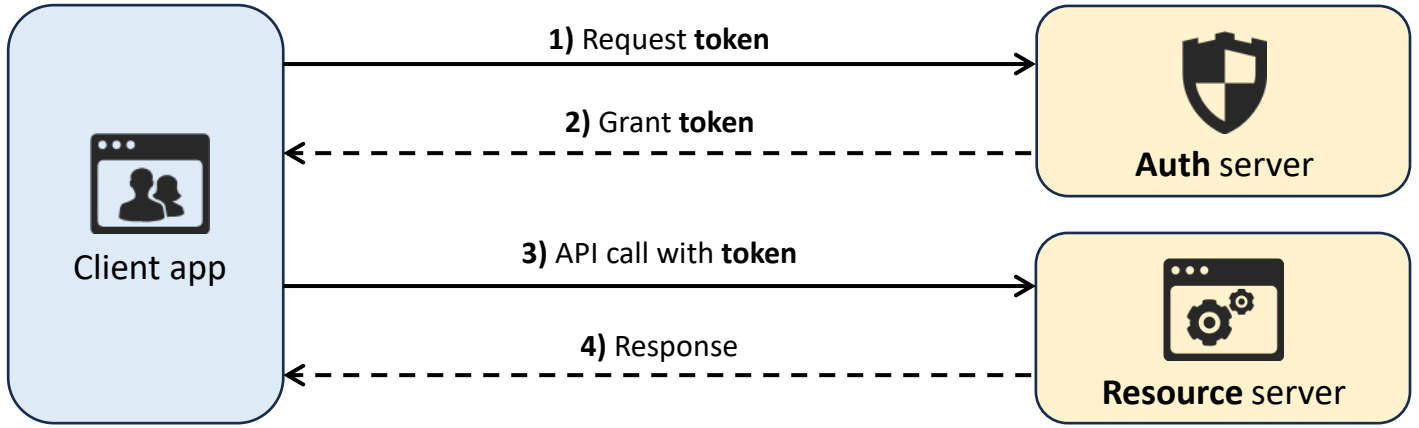
Bearer authentication

- **Bearer authentication** (a form of token-based authentication) uses a **token** instead of a username/password.
  - The client first obtains a token by authenticating with an authorization server.
    - This can be done using using **Basic authentication** or another method.
  - For each API call, the client includes the token in the HTTP Authorization header.
    - e.g., `Authorization: Bearer ya29.a0ARrdaM8`

- The term **bearer** means that anyone who possesses the token can use it.
  - If an attacker steals the token, they can make API calls as if they were the legitimate user.
  - To mitigate against this, **tokens expire** after a set period of time.

- **Advantages:**
  - More secure than **Basic authentication** (no need to transmit the same username/password for every API call).
  - Tokens expire, so a stolen token will only be temporarily valid.

- **Disadvantages:**
  - If a token is stolen, the attacker can access the API until it expires.
  - Tokens need to be refreshed periodically, adding extra complexity to implement.
  - Should only be used with HTTPS.

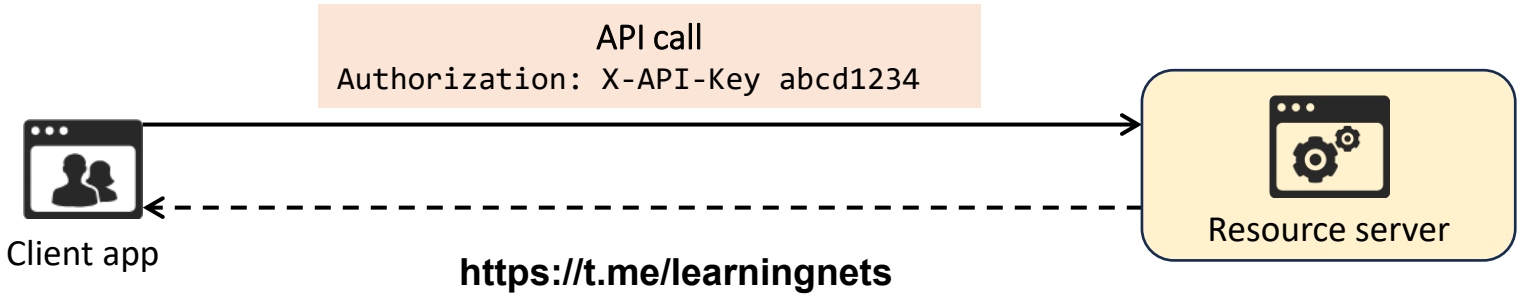


API key authentication

- **API key authentication** uses a **static key** issued by the API provider.
  - The client uses this key in each API call for authentication.
  - Unlike **bearer tokens**, the **API key** is static and remains valid until revoked.

- API keys can be sent in:
  - The HTTP Authorization header (recommended)
  - The URL (e.g., add `?api_key=abcd1234` to the end of the URL)
    - Not recommended! URLs are often logged by web servers, proxies, browsers, etc.
  - A cookie (sometimes used for browser-based APIs).

- **Advantages:**
  - Easier to implement than Bearer authentication (no need to refresh tokens).
  - Good for tracking API usage. Often used by cloud services and third-party APIs.
- **Disadvantages:**
  - If stolen, the key grants full access until revoked.
  - API keys must be rotated manually to maintain security, whereas tokens expire automatically.



OAuth 2.0

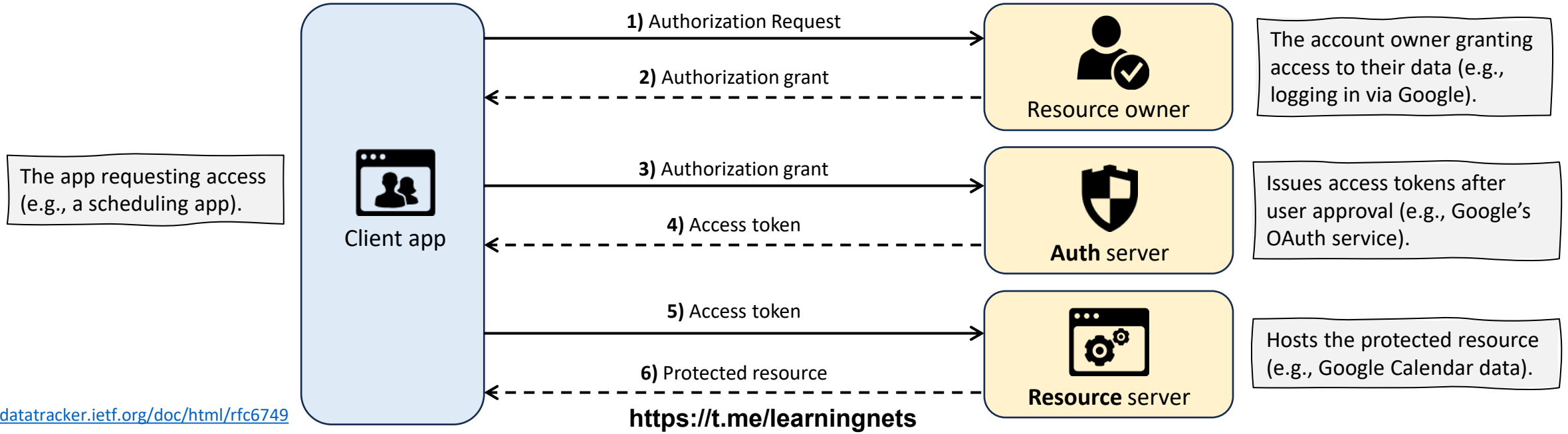
- **OAuth 2.0** is a secure authentication framework that is widely used in modern web applications.
  - It provides *access delegation*, granting third-party applications **limited access** to resources on behalf of the resource's owner.
    - There is no need to share the resource owner's credentials with the third party.

- Examples of OAuth2.0:
  - **Logging in with Google:**
    - Many websites and apps offer the option to log in using your Google account.
  - **Connecting apps to social media accounts:**
    - Many apps can be connected to accounts on social media platforms like Instagram, Facebook, LinkedIn, etc.
  - **Calendar integration:**
    - A third-party tool can be given access to your Google Calendar to check availability and schedule meetings.

OAuth 2.0

- The **OAuth2.0** authentication/authorization process consists of six steps:
  1. The **client app** requests authorization from the resource owner (you) to access the resource (your Google Calendar data).
  2. The **resource owner** grants authorization by logging into their account (e.g., Google account) and giving permission.
  3. The **client app** exchanges the authorization grant for an access token from the auth server.
  4. The **auth server** provides an access token to the client app.
  5. The **client app** sends the access token to the resource server (e.g., Google's server hosting calendar data) to request the resource.
  6. The **resource server** validates the access token and provides the requested resource (calendar data) to the client app.

- The **access token** granted in step 4 functions just like the token used in **bearer authentication**.
  - It grants access to the specified resource within the appropriate scope of access (e.g., read-only access).
  - Access tokens expire after a short period, but OAuth 2.0 uses **refresh tokens** (granted by the Auth server) to obtain new access tokens without requiring the user to log in every time.



https://datatracker.ietf.org/doc/html/rfc6749

Summary

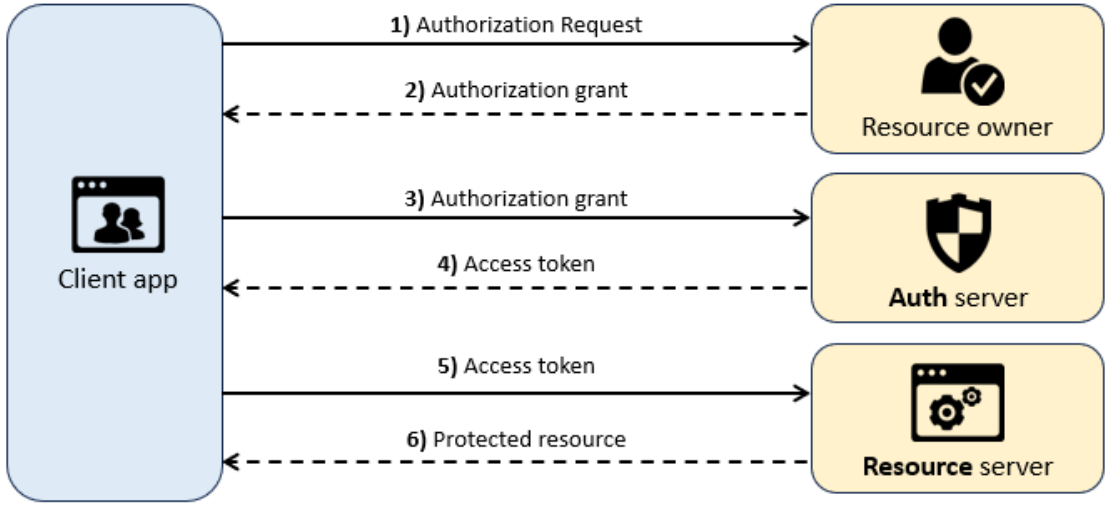
- **REST API authentication** ensures only authorized users or systems can access an API.
  - Without authentication, unauthorized users could access or modify data.

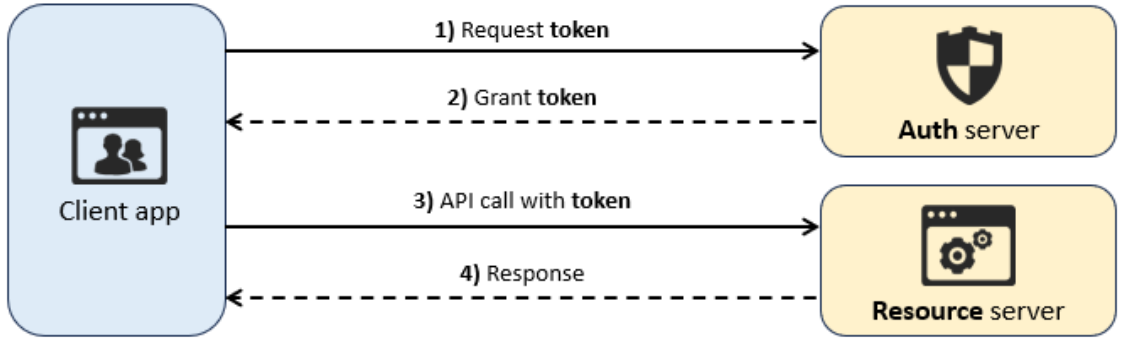
- Four types (methods/schemes) of REST API authentication:
  - **Basic Authentication**
    - Uses a **username and password** encoded in **Base64**, but not encrypted.
    - Requires **HTTPS (TLS)** for security since cleartext credentials are sent with each request.
  - **Bearer Authentication**
    - Uses a **bearer token**, granted by an Auth server, instead of a username/password for authentication.
    - Tokens **expire** after a short time but can be stolen if not protected.
  - **API Key Authentication**
    - Uses a **static key** issued by the API provider for authentication.
    - Good for tracking API usage.
    - Easier to implement but **less secure** since stolen keys remain valid until revoked.
  - **OAuth 2.0**
    - Provides **access delegation**, allowing third-party apps **limited access** to resources.
    - Uses **access tokens** that expire and can be refreshed (with a **refresh token**) without user reauthentication.
    - Four main parties:
      - Resource owner
      - Client app
      - Auth server
      - Resource server

Quiz 1

Which of the following REST authentication methods involve an Auth server issuing tokens? (select two)

- A) OAuth 2.0
- B) Bearer authentication
- C) Basic authentication
- D) API key authentication

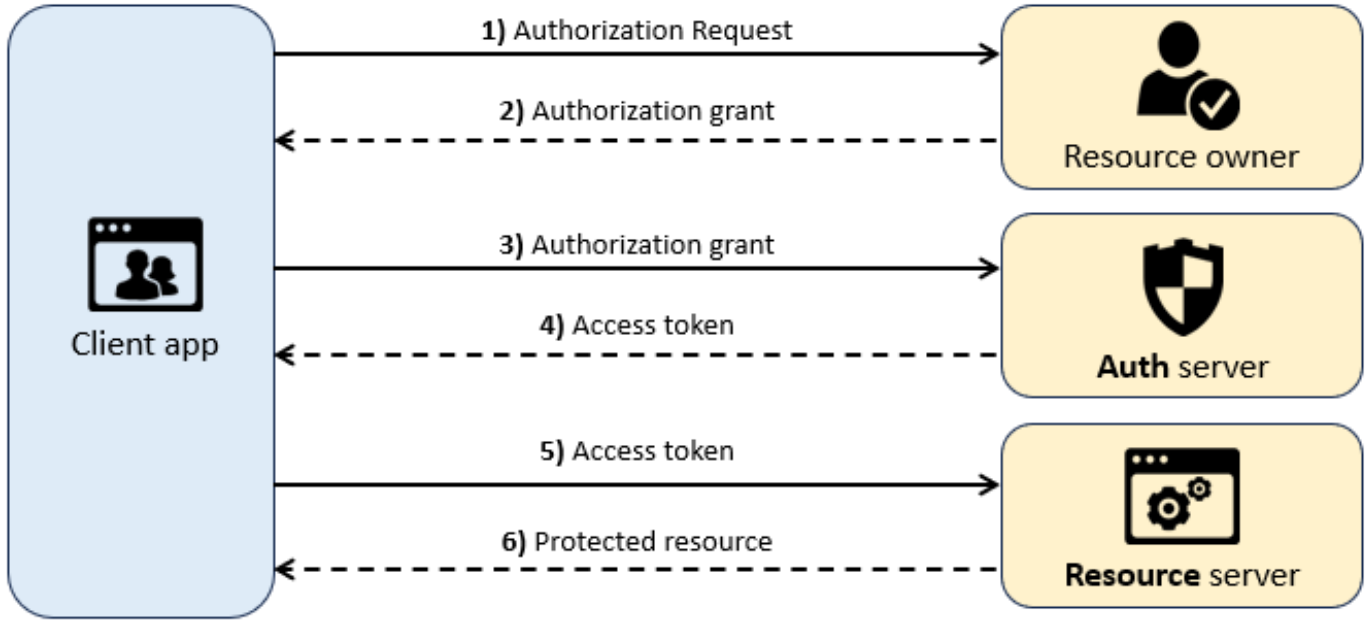




Quiz 2

Which of the following best describes how OAuth 2.0 improves security?

- A) It eliminates the need for authentication when accessing APIs.
- B) It allows third-party apps to access resources without exposing user credentials.
- C) It requires users to manually refresh access tokens.
- D) It stores passwords in cleartext for quick access.



Quiz 3

Which of the following statements are true? (select three)

- A) API keys are more secure than bearer tokens because they expire automatically.
- B) Bearer authentication requires a token issued by an auth server.
- C) OAuth 2.0 enables access delegation, allowing third-party apps limited access to user data.
- D) Basic authentication uses a simple form of encryption to secure API requests.
- E) OAuth 2.0 uses refresh tokens to obtain new access tokens without requiring user reauthentication.
- F) Including API keys in URLs is the most secure way to authenticate API requests.