PIEMONTE
FONDI STRUTTURALI E DI INVESTIMENTO EUROPEI 2014/2020
fondo sociale europeo
ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

# RESTful API: Principali Metodi di Autenticazione

Pier Paolo Pittavino

in collaborazione con:
ISTITUTI TECNICI SUPERIORI
Ministero dell'Istruzione
UNIONE EUROPEA
REGIONE PIEMONTE
per una crescita intelligente, sostenibile ed inclusiva
www.regione.piemonte.it/europa2020
INIZIATIVA CO-FINANZIATA CON FSE

## Definizione 0.1: Autenticazione

È il processo che verifica l'identità di un utente in modo da poter correttamente permettere o negare l'accesso a risorse condivise e protette

## Definizione 0.2: Autorizzazione

È il processo che consente l'accesso alle risorse solamente a coloro che hanno i diritti di usarle (**DOPO** il processo di autenticazione)

# Authentication vs Authorization

**Authorization**

What you can do

**Authentication**

Who you are

# Metodi di Autenticazione

Esistono moltissimi metodi (spesso proprietari) per autenticarsi a delle Rest API, spesso però sono varianti di pochi approcci principali.
Ne vedremo in particolare 4 tra i più usati nel mondo delle RestFul API e dei microservizi:

1. HTTP Authentication Schemes (Basic  Bearer)
2. API Keys
3. OAuth (2.0)
4. OpenID Connect

# 1. HTTP Auth Schemes (Basic & Bearer)

Il protocollo HTTP stesso definisce i suoi proprio schemi di autenticazione, nel mondo delle API se ne usano in particolare 2:

1. **Basic Authentication:** Il metodo più semplice ma sostanzialmente insicuro
2. **Bearer Authentication:** schema di autenticazione HTTP che prevede l'uso di token di sicurezza detti appunto *Bearer tokens*

# 1.1 HTTP Basic Authentication

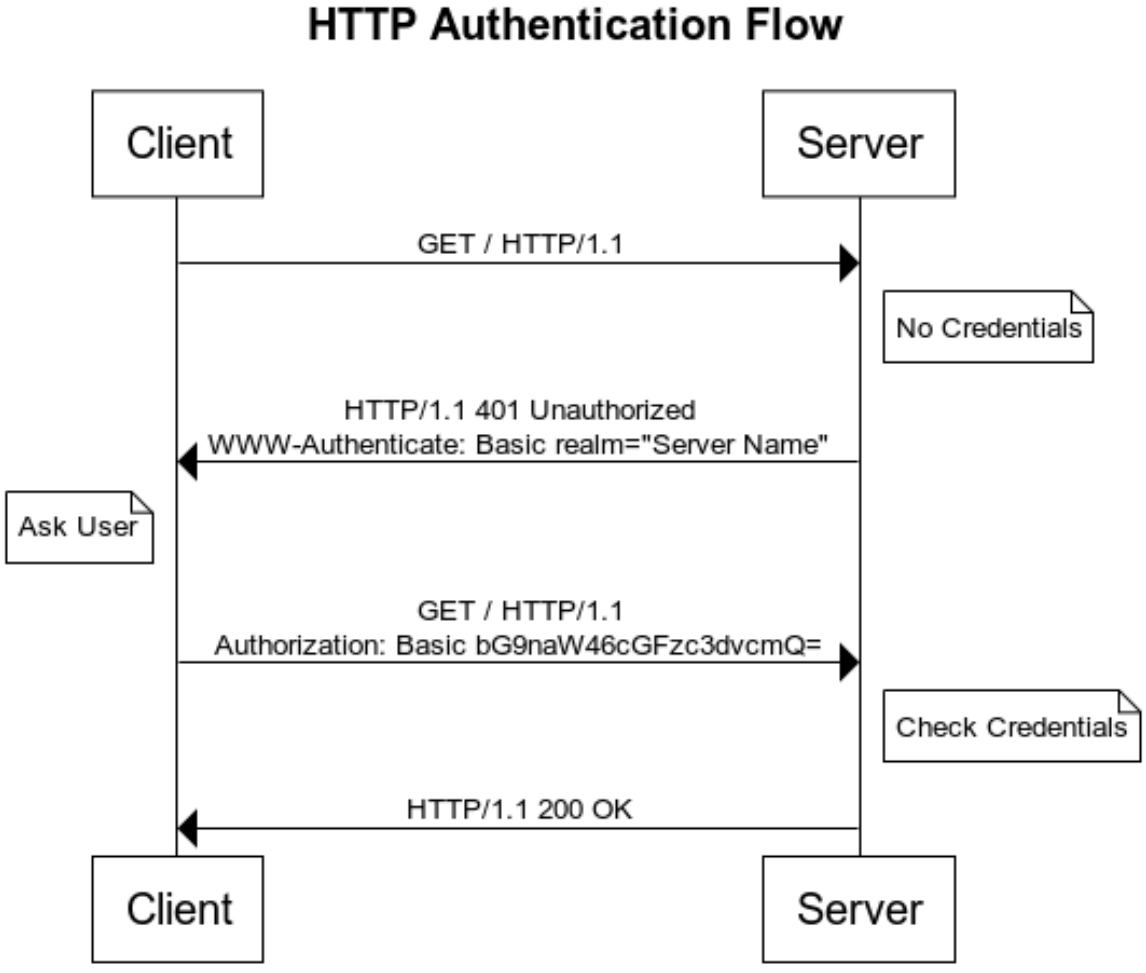
Non raccomandato perché sostanzialmente insicuro.
Il richiedente inserisce direttamente nell'header della richiesta *username:passowrd* dopo averlo encodato in *base64*

**Esempio 0.1: Header - Basic Auth**

```
Authorization: Basic bG9sOnNlY3VyZQ==
```

**Deve sempre essere usata con HTTPS (SSL) !!**

# 1.1 HTTP Basic Authentication



# 1.2 HTTP Bearer Authentication

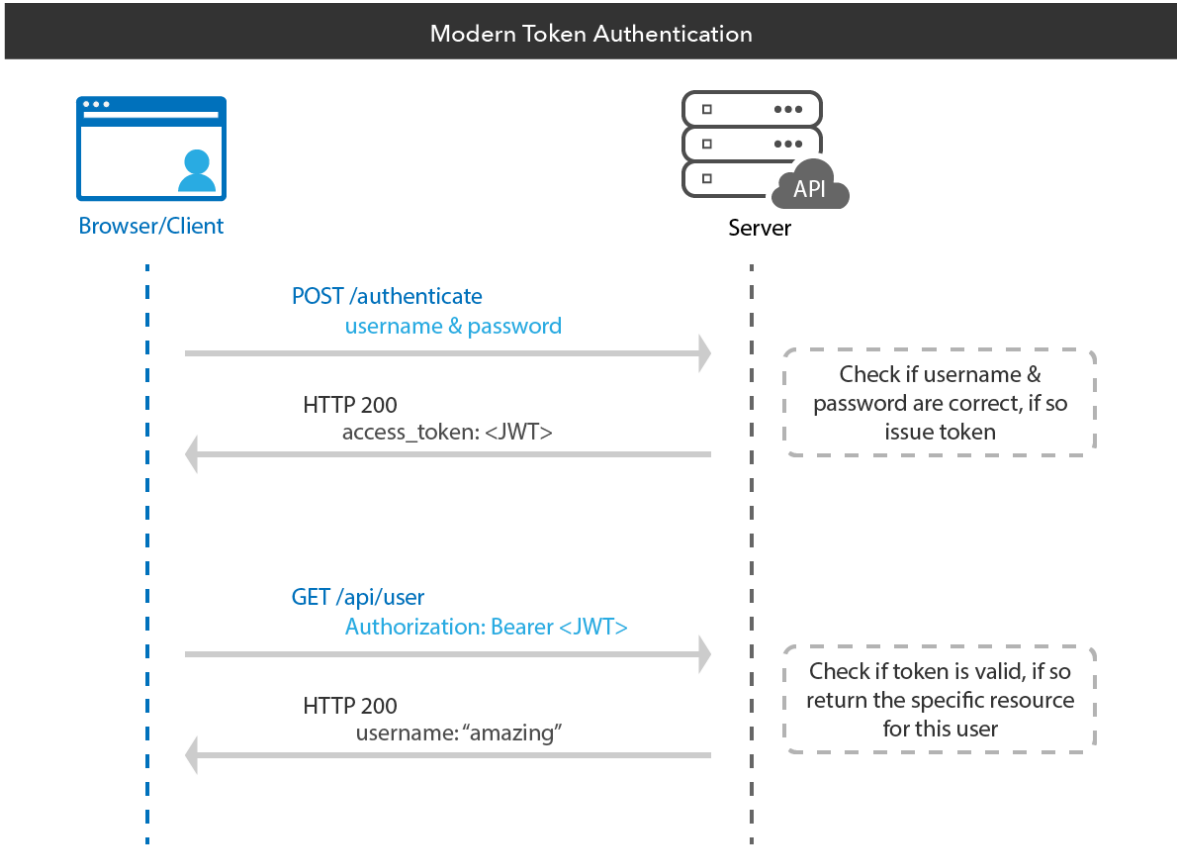
*Bearer* significa *portatore/portatrice*
La richiesta *portatrice* di un determinato (spesso complesso) token nel suo header, solitamente fornito dal server in risposta ad una richiesta di login, avrà accesso a determinate risorse o url.

**Esempio 0.2: Header - Bearer Auth**

Authorization: Bearer < *token* >

**Deve sempre essere usata con HTTPS (SSL) !!**

# 1.2 HTTP Bearer Authentication



# 2. API Keys

Molto diffuse ed una sorta di standard per accedere a delle API Rest, tuttavia non particolarmente sicuro.
Una chiave univoca (generata randomicamente dal server) viene associata ad un utente la prima volta che questi ne fa richiesta, e viene quindi utilizzata le volte successive per identificare l'utente.
Spesso inserite come query string direttamente nell'url (Da NON fare!!) è più opportuno inserirlo in un header di autorizzazione:

**Esempio 0.3: Header - API Key**

```
Authorization: Apikey 1234567890abcdef
```

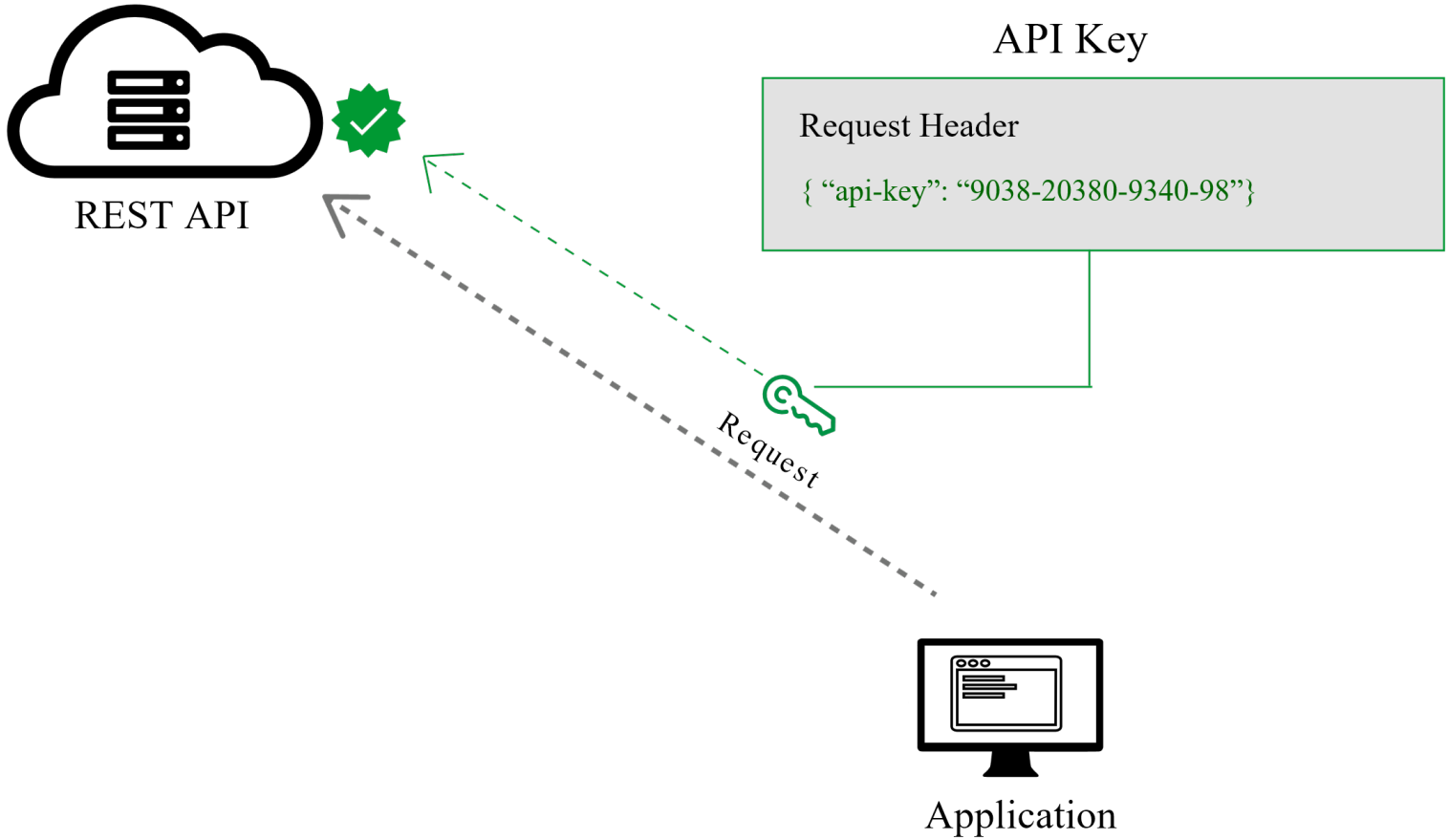
# 2. API Keys

Le *API keys* sono semplici, molto usate da diversi servizi cloud soprattutto quando si tratta di **servizi in sola lettura**

Tuttavia sono trasmesse in chiaro sulla rete come le precedenti autenticazioni per cui **devono sempre essere usate con HTTPS (SSL) !!**

Oltre che nell'*authorization header* e nel *query string*, talvolta vengono usate nel *body* o dentro una *basic auth* o in *header custom*

# 2. API Keys


API Key
Request Header
{ “api-key”: “9038-20380-9340-98”}
REST API
Request
Application

# 3. OAuth (2.0)

La versione 2 di questa specifica è più semplice delle precedenti
Nelle implementazioni più comuni fa uso di uno (o due) tokens:

- **access token:** utilizzato come un *API key* consente all'applicazione che lo utilizza di accedere ad i dati dell'utente ed eventualmente può avere una scadenza
- **refresh token:** utilizzato opzionalmente per richiedere nuovi token di accesso quando questi sono scaduti

OAuth 2.0 combina *autenticazione* ed *autorizzazione* ed è la scelta più matura per identificare utenti e garantire i permessi corretti

# 3. OAuth (2.0): building block

oAuth 2 divide i ruoli e prevede per ognuno un compito ben definito:

- **Resource Owner**: è il proprietario dell'informazione esposta via HTTP.
- **Client**: è l'applicazione che richiede l'accesso alla risorsa HTTP.
- **oAuth 2 o Authorization Server**: è il modulo che firma e rilascia i token di accesso e, se necessario, richiede la login al Resource Owner
- **Resource Server**: è il server che detiene l'informazione esposta via HTTP

# 3. OAuth (2.0): grant type

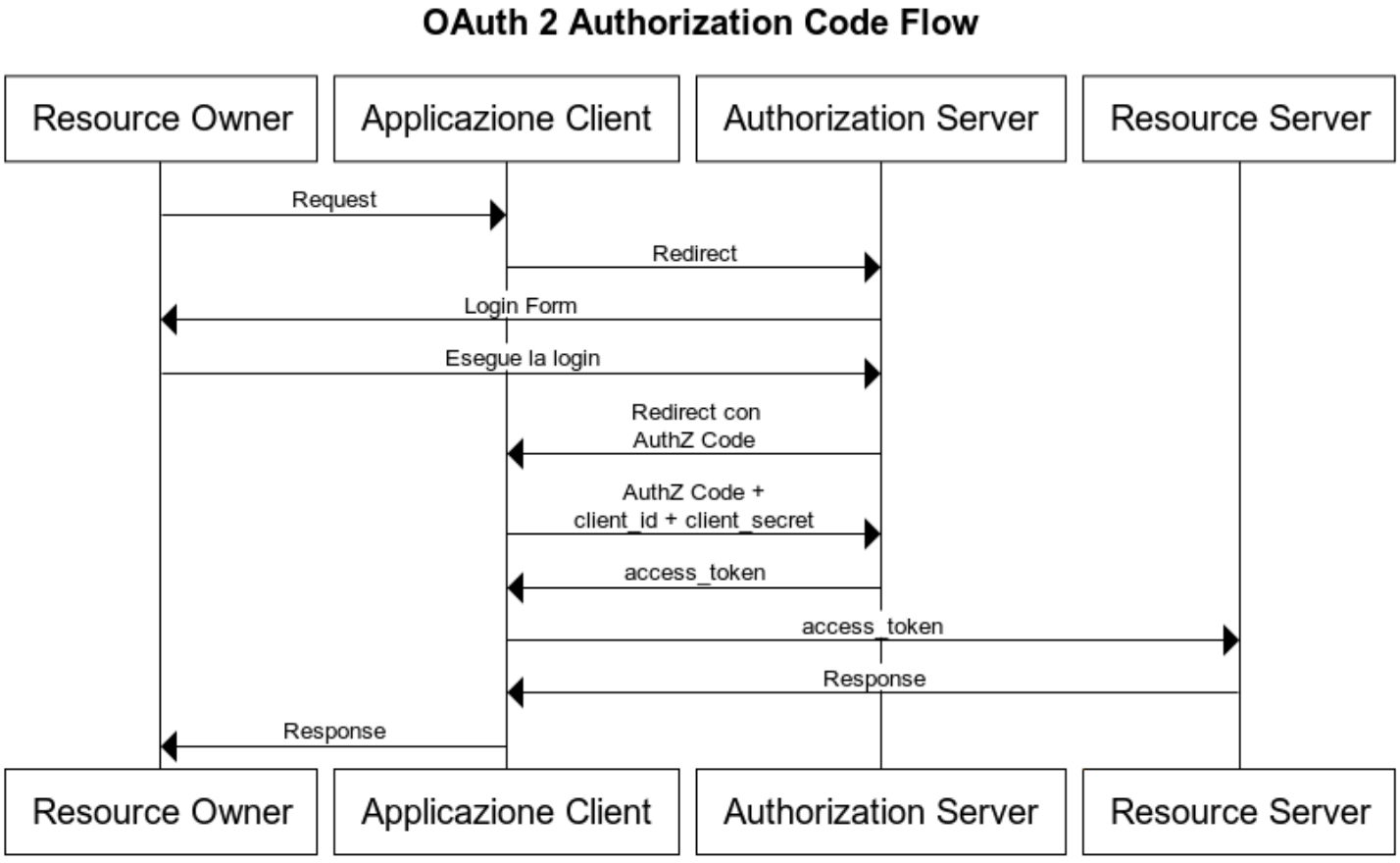
Un Grant Type è il processo da seguire per ottenere il cosidetto Authorization Grant, ovvero la prova inoppugnabile dell'avvenuta autorizzazione da parte del Resource Owner, il titolare dell'informazione, a cui l'applicazione Client (colei che chiede l'accesso alla risorsa ad esempio un app) sta cercando di accedere. 4 diverse modalità:

1. **Authorization Code Grant Type**
2. Implicit Grant Type
3. Resource Owner Password Credentials Grant Type
4. Client Credentials Grant Type (tra applicazioni server-side)

# 3.1 Authorization Code Grant Type

1. Il Resource Owner accede all'applicazione Client
2. L'applicazione Client esegue il redirect all'Authorization Server (oAuth 2 Server)
3. L'Authorization Server chiede al Resource Owner di autenticarsi
4. A valle della login andata a buon fine, l'Authorization Server consegna all'applicazione Client l'Authorization Code
5. L'applicazione Client riconsegna all'oAuth 2 Server l'Authorization Code appena ricevuto, unitamente a client_id e client_secret (potremmo definirle l'username e la password dell'applicazione Client)
6. L'Authorization Server consegna all'applicazione Client un access_token per consentire all'applicazione Client di venire autorizzato presso il Resource Server

# 3.1 Authorization Code Grant Type



# 3.1 Authorization Code Grant Type

Il Resource Server non ha bisogno di contattare l'Authorization Server per verificare la validità dell'access_token perché solitamente il token è autoreferenziale, ovvero firmato con crittografia asimmetrica dall'Authorization Server (JWT).
Il Resource Server deve quindi avere la chiave pubblica dell'oAuth 2 Server per validare il token.

# 3. OAuth (2.0): tokens

Solitamente si utilizza un access token (e un refresh token) autoreferenziale, ovvero firmato digitalmente:

- può essere validato dal Resource Server senza dover chiedere una verifica all'Authorization Server
- La chiave privata è in possesso solo dell'Authorization Server
- la chiave pubblica è in possesso del/dei Resource Server

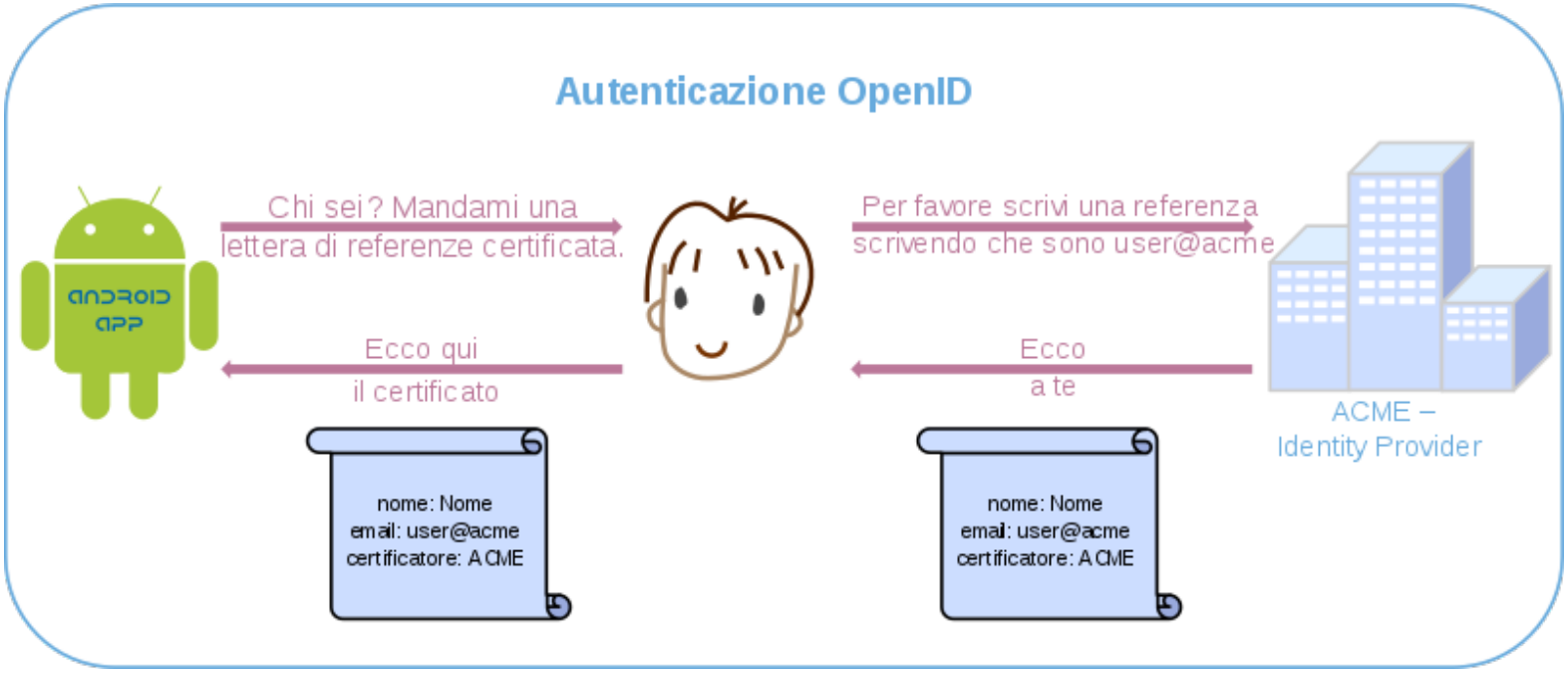
# 4. OpenID Connect

- È un layer aggiuntivo sopra il protocollo OAuth2
- Consente ai client di verificare l'identità di un utente finale nonché di ottenere informazioni di base tramite un'API HTTP RESTful, utilizzando JSON come formato dati
- è un protocollo di autenticazione decentralizzato ideato per autenticarsi in un servizio (service provider) riutilizzando account già esistenti da altri siti web (identity provider). Lo scopo è di evitare che ogni utente necessiti di registrare un account in ogni fornitore di servizi

# 4. OpenID Connect

1. Un'applicazione richiede all'utente di effettuare l'accesso
2. L'utente sceglie uno fra gli identity provider a disposizione o comunque ne specifica uno (che da qui in poi chiameremo "ACME")
3. L'utente è reindirizzato al sito di ACME
    1. L'utente effettua il login nel sito di ACME
    2. ACME eventualmente chiede all'utente quali informazioni serve trasmettere al servizio (solo l'e-mail ad esempio)
    3. ACME rilascia all'utente un certificato che può essere utilizzato per verificare che egli si tratti effettivamente di un'utenza ACME , L'utente ritorna automaticamente al sito del servizio di origine consegnandogli il certificato e le informazioni personali
4. L'utente è ora autenticato

# 4. OpenID Connect



ITS
TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

# GRAZIE!

POR Piemonte
FSE 2014-2020