

# Digital Identity Management

Iniziamo a parlare dei **protocolli proposti dalla FIDO2 Alliance**, i quali **autenticano gli utenti sulla base della crittografia a chiave pubblica**, ovvero: ad ogni utente è assegnato una coppia di chiavi (una chiave pubblica e la corrispondente chiave privata) e per essere autenticati, gli utenti devono eseguire un'operazione crittografica. Tipicamente, gli utenti devono firmare con la chiave privata una serie di dati, al fine di provare che essi sono i proprietari della chiave privata. A questo punto, il Server di autenticazione può verificare la firma con la corrispondente chiave pubblica di cui mantiene una copia. I protocolli della FIDO2 Alliance vengono utilizzati in due scenari:

1. Lo scenario di **autenticazione senza password**, in cui l'utente è autenticato semplicemente attraverso le operazioni di firma, che abbiamo nominato sopra;
2. Lo scenario di **protocolli di autenticazione a due o più fattori**, in cui l'utente è autenticato sulla base di una password e tramite l'autenticazione basata su crittografia a chiave pubblica.

Sorge a questo punto spontanea la domanda: **“Quali sono i vantaggi e perchè questi protocolli sono più sicuri rispetto agli altri protocolli visti fino a questo momento?”** Abbiamo visto, che uno dei principali problemi dei protocolli visti fino a questo momento, è che le **credenziali** (come lo username e la password) possono essere:

- **rubate** tramite attacchi di phishing;
- oppure possono essere **riutilizzate** per autenticare l'utente presso siti diversi, rispetto al sito in cui sono state rubate le credenziali dell'utente dall'attaccante.

Con l'autenticazione, prevista dai protocolli della FIDO2 Alliance, **il problema appena sopra citato non può verificarsi, in quanto la coppia di chiavi <pubblica-privata> che viene assegnata all'utente, sono legate al sito Web per cui sono state registrate** → questo comporta, che se anche l'attaccante riuscisse ad ottenere una copia delle credenziali e tentasse di utilizzare la coppia di chiavi <pubblica-privata> in altri siti Web, l'autenticazione fallirebbe → in questo modo gli attacchi di phishing e di replicate attacks, in cui l'attaccante cerca di ottenere una copia delle chiavi <pubblica-privata> dell'utente per autenticarsi in diversi siti, vengono bloccati.

Inoltre, con questi protocolli abbiamo un **maggior rispetto della privacy degli utenti**, in quanto rispetto ai sistemi di autenticazione basati su password (dove abbiamo che l'username, la password e l'email devono essere mantenuti dal service provider, il quale verifica l'identità degli utenti), con i protocolli della FIDO2 Alliance abbiamo che l'unica informazione mantenuta dal Server, che si occupa di verificare l'identità dell'utente, è la chiave pubblica dell'utente ed il suo username.

**“Quali sono i principali attori coinvolti nell'implementazione di questi protocolli?”** I principali attori sono:

- **L'utente** che deve autenticarsi al servizio online che supporta questo tipo di autenticazione;
- Il **relying party**, che è l'organizzazione (o in generale il service provider) responsabile della registrazione e dell'autenticazione dell'utente;
- La **Client platform**, la quale comprende:
  - l'applicazione che l'utente utilizza per registrarsi e autenticarsi presso il relying party;
  - il dispositivo che l'utente utilizza (come per esempio lo smartphone o il PC).
- Gli **autenticatori**, che sono la componente principale per implementare tali protocolli, in quanto sono i responsabili di:
  - generare la coppia di chiave <pubblica-privata> utilizzata dall'utente per autenticarsi;
  - memorizzare la coppia di chiavi in modo sicuro;
  - eseguire le operazioni di firma, utilizzate per autenticare l'utente.

Da notare il fatto, che vi possono essere due tipologie di autenticatori:

1. **autenticatore esterno** → il quale è esterno alla piattaforma che l'utente utilizza per autenticarsi (un esempio di autenticatore esterno è la chiavetta USB);
2. **autenticatore embedded nel dispositivo** (come per esempio Windows Hello).

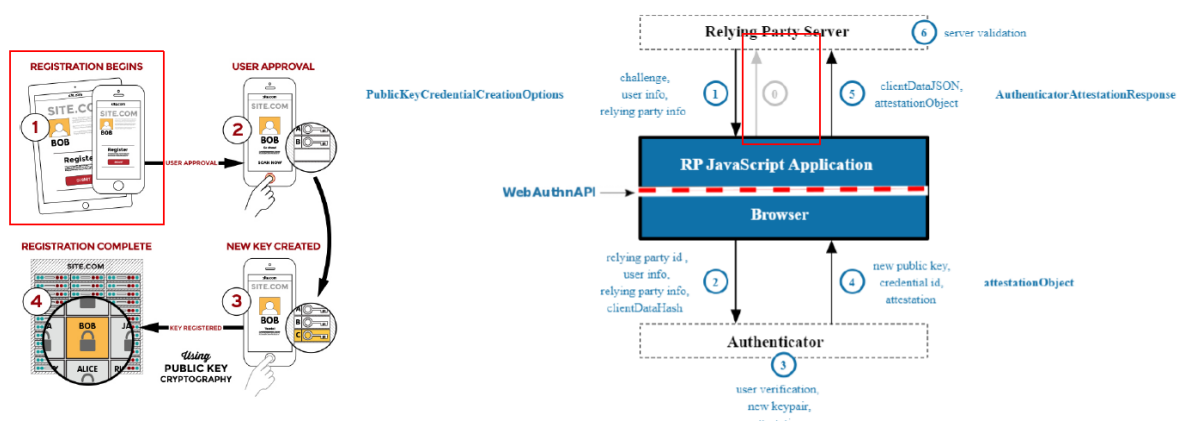
Solitamente i produttori degli autenticatori associano ad ogni autenticatore un identificativo univoco, che consente al Server del relying party, che deve fare l'autenticazione dell'utente, di recuperare le proprietà di sicurezza garantite dall'autenticatore rispetto alle chiavi utilizzate dall'utente per autenticarsi e rispetto alle operazioni di cifratura eseguite per l'autenticazione. Inoltre,

solitamente agli autenticatori (soprattutto quelli esterni) sono associati una coppia di chiavi <pubblica-privata>, dette **chiavi di attestazione**, che servono per provare al Server del relying party il fatto che l'autenticatore garantisca determinate proprietà di sicurezza.

Vi sono poi due **specifiche** per implementare i protocolli della FIDO2 Alliance, ovvero:

1. **WebAuth** → un'API JavaScript che viene integrata nei browser e nelle piattaforme per consentire il supporto dell'autenticazione FIDO;
2. **CTAP2** → un protocollo per interagire con autenticatori esterni (chiavi di sicurezza FIDO, dispositivi mobili) per l'autenticazione su browser e sistemi operativi abilitati a FIDO2 tramite USB, NFC o BLE per un'esperienza di autenticazione senza password, a secondo fattore o a più fattori.

Adesso vediamo come effettivamente vengono implementati questi protocolli:



Come per qualsiasi protocollo di autenticazione, abbiamo una fase iniziale di registrazione, in cui:

- l'utente può avere già un account presso il sito del relying party presso il quale si vuole autenticare;
- oppure l'utente crea un nuovo account all'inizio della procedura di registrazione. Nel momento in cui l'utente crea un nuovo account, l'obiettivo è di creare una coppia di chiavi <pubblica-privata> e di passare la chiave pubblica al Server del relying party, in modo tale che si possa utilizzare tale chiave pubblica per verificare l'identità dell'utente durante la fase di registrazione.

Vediamo meglio ogni fase ed in particolare abbiamo:

1. Nella prima fase, se l'utente aveva già un account, quest'ultimo (ovverosia l'utente) può collegarmi tramite il suo username e la propria password, altrimenti viene creato un nuovo username nel momento in cui inizia la cerimonia di registrazione. Successivamente, abbiamo che il Server del relying party invia delle informazioni all'applicazione Client che viene eseguita nel browser dell'utente. In questo caso, quindi, la Client Platform è rappresentata dal browser dell'utente e dall'applicazione Client eseguita all'interno del browser → in particolare, abbiamo che le informazioni inviate dal Server del relying party all'applicazione Client sono:

- una challenge → ovverosia un valore completamente casuale, utilizzato per evitare degli attacchi di tipo replay, in cui l'attaccante può riutilizzare le chiavi per autenticarsi (la challenge viene passata come prima informazione);
- viene specificato chi è il relying party → quindi vengono inviate informazioni relative al nome e al dominio dell'organizzazione che sta registrando l'identità dell'utente;
- informazioni sull'utente → quali per esempio: il nome da mostrare quando l'utente si collega e lo user id, come per esempio lo username dell'utente;
- informazioni relative al tipo di chiave che deve essere generata per autenticare l'utente;
- quale tipo di autenticatore può essere adottato per svolgere le operazioni crittografiche. Per esempio, se abbiamo un autenticatore incluso nella piattaforma Client, allora parliamo di autenticatore embedded, altrimenti possiamo avere un autenticatore esterno;
- un'attestazione da parte dell'autenticatore (utilizzato dall'utente) che certifichi che l'autenticatore è fidato. In particolare, abbiamo che l'attestazione può essere:
  - **direct** → l'autenticatore deve fornire una prova delle proprietà di sicurezza che può garantire;
  - **none** → non viene richiesta alcuna attestazione all'autenticatore;
  - **self-attested** → l'autenticatore di auto-certifica che garantisce determinate proprietà di sicurezza.
- il tempo in millisecondi per cui una richiesta di autenticazione rimane valida. Trascorso tale tempo, se l'utente non agisce, la registrazione fallisce.



Tutte queste informazioni vengono passate in un oggetto chiamato **PublicKeyCredentialCreationOptions** dal relying party Server all'applicazione Client.

2. Alcune di queste informazioni contenute nel `PublicKeyCredentialCreationOptions` vengono passate dall'applicazione Client all'autenticatore ed in particolare vengono passate le informazioni:
  - relative al relying party che deve autenticare l'utente;
  - relative all'utente, come per esempio lo user-id associato all'utente;
  - l'hash di alcune informazioni relative al Client. In particolare, tali informazioni contengono tre parametri:
    - la **challenge**, che il relying party Server aveva inviato all'applicazione Client;
    - l'**origine**, ovvero il dominio associato al relying party;
    - una **stringa**, che rappresenta il tipo di operazione svolta (in questo caso è un'operazione di `WebAuthn.create`, ovvero di creazione di una nuova credenziale).
3. Dopodiché, abbiamo che l'utente deve dare il proprio consenso che vuole continuare con l'operazione di registrazione di una nuova credenziale. Se si sta utilizzando lo smartphone, questa operazione può corrispondere a specificare un pass-key (ovvero inserire un PIN) oppure utilizzare Face-ID → in questo modo, l'autenticatore capisce che l'utente vuole procedere con l'operazione di registrazione.
4. A questo punto, l'autenticatore genera:
  - una nuova coppia di chiavi <pubblica-privata>;
  - l'attestazione, che di fatto è una firma fatta sulla chiave pubblica appena generata e l'identificativo univoco assegnato all'utente.

In questo modo, quando il relying party Server quando riceverà l'attestazione da parte dell'applicazione Client, può andare a recuperare (tramite l'identificativo univoco) le proprietà di sicurezza garantite dall'autenticatore. Tutte queste informazioni, poi sono contenute in un `attestationObject` e vengono passate dall'autenticatore a nuovamente l'applicazione Client. Nell'`attestationObject`, quindi, abbiamo:

- la chiave pubblica appena creata;
- un identificativo della chiave, chiamato “credential-id”;
- l’attestazione, ovverosia la firma, generata dall’autenticatore.

In particolare, abbiamo che le informazioni contenute nell’attestationObject sono:

- il parametro “fmt” → indica il formato dell’attestazione, che è stata generata dall’autenticatore;
- l’attestation statement → il quale include:
  - la firma;
  - specifica l’algoritmo che l’autenticatore ha utilizzato per generare la firma;
  - un certificato della chiave pubblica, che consente poi al relying party Server di verificare la firma generata con la chiave privata di attestazione associata all’autenticatore.
- l’authenticator-Data → il quale contiene un identificativo del relying party che deve autenticare l’utente e gli Attested Credential Data, i quali a loro volta contengono:
  - la chiave pubblica appena generata;
  - l’identificativo della chiave pubblica;
  - l’identificativo dell’autenticatore, che ha generato la chiave pubblica.

5. Infine, abbiamo che l’attestationObject viene restituito all’applicazione Client e a questo punto, l’applicazione Client genera un nuovo oggetto chiamato **“AuthenticatorAttestationResponse”**, che contiene l’attestationObject e i dati sul Client (che sono gli stessi che sono stati passati dall’applicazione Client all’autenticatore durante la 2° fase). In particolare, quindi, nell’AuthenticatorAttestationResponse abbiamo:

- il ClientData JSON → consente al relying party di associare la chiave pubblica appena generata con l’applicazione Client;
- l’attestationObject → il quale ricordiamo contiene la chiave pubblica, che il relying party Server salverà in un Database locale insieme al suo identificativo (ovverosia il credential-id) e poi contiene l’attestazione. Il relying party Server, quindi, va a recuperare la chiave pubblica mediante il suo certificato incluso nell’attestation statement dell’attestationObject e

verifica la firma fatta dall'autenticatore. Se la firma è valida, allora, è sicuro che la chiave pubblica è stata generata da un autenticatore fidato, che garantisce determinate proprietà di sicurezza.

A questo punto abbiamo una registrazione completa, dato che abbiamo un'associazione tra l'utente e la sua chiave pubblica mantenuta dal relying party Server.

---

Nella fase di autenticazione, invece, all'utente viene mostrata una prima pagina di login, in cui l'utente deve inserire il proprio username. A questo punto, il relying party Server genera un altro oggetto contenente informazioni relative:

- alle chiavi che l'utente ha registrato e che quindi può usare per l'autenticazione;
- agli autenticator che l'utente può utilizzare per autenticarsi;
- una challenge, che serve per prevenire attacchi di tipo replay.

A questo punto, queste informazioni vengono passate dall'applicazione Client all'autenticatore e quest'ultimo deve eseguire l'operazione di firma, che attesta che l'utente possiede la chiave privata corrispondente alla chiave pubblica registrata durante la fase di registrazione. La firma viene generata sulla base di due informazioni:

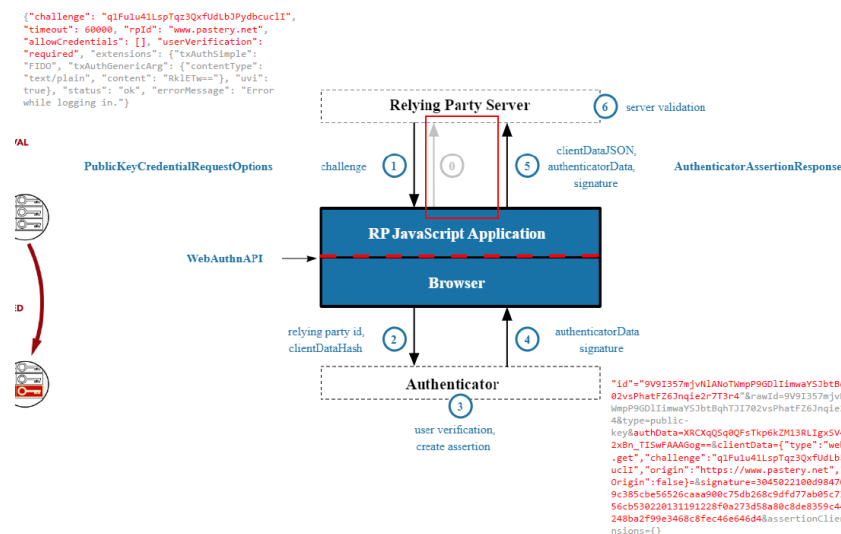
1. l'hash del ClientData, che contiene praticamente le stesse informazioni riguardanti il Client visto nella fase di registrazione, ovvero contiene:
  - a. l'origine, ovvero il dominio associato al relying party;
  - b. una stringa, che rappresenta il tipo di operazione (che in questo caso l'operazione è WebAuthn.get);
2. il relying party id.

L'autenticatore, quindi, recupera la chiave privata associata all'utente, genera la firma sull'hash del Client e sui dati relativi all'autenticatore (i quali sono simili a quelli visti nella fase di registrazione, ad eccezione del fatto che in questo caso non c'è la chiave pubblica). Una volta che ha generato la firma, l'autenticatore rimanda indietro all'applicazione Client:

- gli authenticator Data, che non contengono la chiave pubblica;
- la firma generata sugli authenticator Data e sulla hash del ClientData;
- l'id della credenziale;

- un parametro opzionale, chiamato “User handler”, che è un identificativo dell'utente.

Tutto questo, infine, viene passato dall'applicazione Client al Server..



Sorge a questo punto spontanea la domanda: **“L'autenticazione senza password e a due fattori basata su WebAuthn API è sicura?”** Sono state identificate quattro categorie di vulnerabilità:

1. la prima vulnerabilità (che è anche quella più diffusa) è la No TLS binding, la quale è legata al fatto, che benché l'autenticazione della vittima e l'autenticazione dell'attaccante avvengano su due canali TLS diversi, l'implementazione dei siti vulnerabili a questa prima vulnerabilità, non vanno a controllare che la risposta che il Client della vittima invia al relying party sia inviata sullo stesso canale TLS utilizzato inizialmente per la richiesta di autenticazione;
2. la seconda vulnerabilità, invece, è legata all'utilizzo della challenge. In particolare, la challenge viene inviata al relying party Server sia nella fase di registrazione sia nella fase di autenticazione e viene utilizzata per evitare, che i dati dell'autenticatore del Client vengano riutilizzati per autenticarsi in altre sessioni di autenticazione da parte di altri utenti. Il problema sta nel fatto, che nell'implementazione di alcuni siti Web, una volta che la challenge è stata utilizzata durante il processo di autenticazione, non veniva invalidata;
3. la terza vulnerabilità e la quarta vulnerabilità riguardano il fatto, che il relying party una volta che riceve i dati relativi all'autenticatore e al Client, non va a verificare che lo user-id associato alla chiave sia dello stesso utente che ha iniziato il processo di autenticazione.

---

## Protocolli di accesso singolo

Quando abbiamo parlato dei sistemi di autenticazione basati su password, abbiamo visto che uno dei principali problemi è per ciascun account a cui l'utente si deve autenticare, l'utente deve ricordare una coppia di <username-password> → questo, naturalmente, porta l'utente ad adottare dei comportamenti (per ricordarsi l'username e la password associati ad ogni account) che lo rendono vulnerabile ad attacchi. Oltretutto, dal punto di vista del service provider, quest'ultimo (ovverosia il service provider) deve implementare il servizio e deve mantenere una serie di informazioni relativi all'account dell'utente, fornite dall'utente stesso durante la fase di registrazione → quindi, oltre al costo di implementare del sistema di autenticazione basato su password, il service provider si espone anche a rischi di attacchi, che possono compromettere l'accesso alle informazioni con conseguenti danni di reputazione al service provider e correlati danni economici.

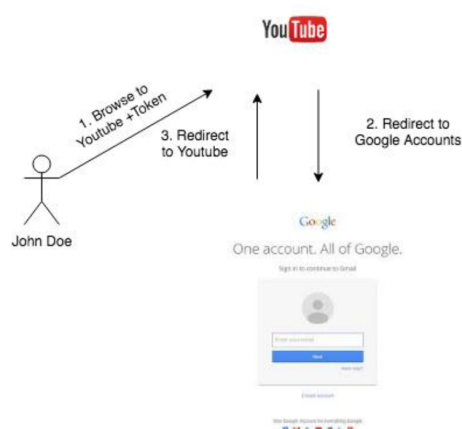
Una possibile soluzione, per risolvere questo problema, è di **sollevare il service provider dall'implementazione del sistema di autenticazione degli utenti e quindi delegare questa parte ad una piattaforma di gestione delle identità digitali**. In questo modo, è la piattaforma che è responsabile di assegnare le credenziali all'utente e per verificare che l'utente sia effettivamente chi dice di essere → otteniamo, quindi, una **soluzione centralizzata**, che oltretutto ci permette di estendere le informazioni che utilizziamo per autenticare un utente, perchè in generale l'utente può autenticarsi non solo con username e password, ma anche con una serie di informazioni che lo identificano in maniera univoca e che, quindi, costituiscono la sua identità digitale → l'identità digitale di un utente, quindi, viene generata presso un servizio di piattaforma d'identità digitale e poi può essere per provare ai service provider l'identità dell'utente.

Questo ci permette di implementare un **servizio di Single Sign On**, perchè un utente può utilizzare solamente un unico set di credenziali (ovverosia solamente quelle associate all'utente dall'identity provider) e autenticarsi una sola volta all'identity provider e una volta autenticato può accedere ad altri servizi online (senza doversi ri-autenticare). Nei servizi di Single Sign On abbiamo tre attori fondamentali:

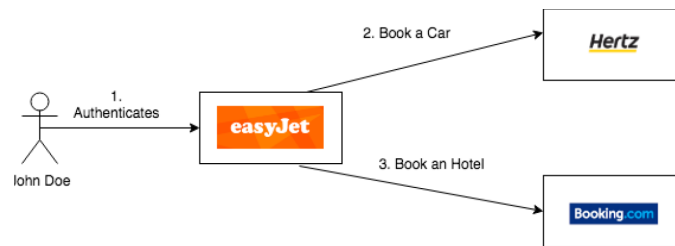
1. l'**utente** di cui vogliamo verificare l'identità;
2. l'**identity provider**, che si occupa di rilasciare l'identità digitale all'utente e di verificarne l'originalità;
3. il **service provider (o relying party)**, che delega il processo di autenticazione all'identity provider e di conseguenza il service provider si fida del processo di

autenticazione fatto dall'identity provider, per garantire l'accesso e l'utilizzo all'utente delle sue risorse e servizi online.

Un esempio di servizio di Single Sign On è quando ci autenticiamo, presso un servizio, attraverso le credenziali di Google o di Facebook. **“Come riusciamo ad autenticarsi tramite le credenziali di Google o di Facebook?”** Essenzialmente, all'utente vengono rilasciate un set di credenziali da Google o da Facebook (o comunque da un altro identity provider) e l'utente si autentica con il set di credenziali presso l'identity provider ed una volta che si è autenticato con successo, l'utente tenta di accedere ad un altro servizio. Quando tenta di accedere ad un altro servizio, l'utente viene ridiretto all'identity provider (nel nostro caso Google o Facebook) e se l'utente si è già autenticato, l'identity provider rilascia all'utente un'asserzione (ovverosia una prova) che dimostra che l'utente è già stato autenticato con successo e che quindi l'utente può “mostrare” (ovverosia mostra l'asserzione) al servizio a cui voleva accedere. Il servizio, a questo punto, controlla la validità dell'asserzione rilasciata dall'identity provider e se l'asserzione è valida, allora il servizio concede l'accesso all'utente. Vediamo un esempio di servizio di Single Sign On:



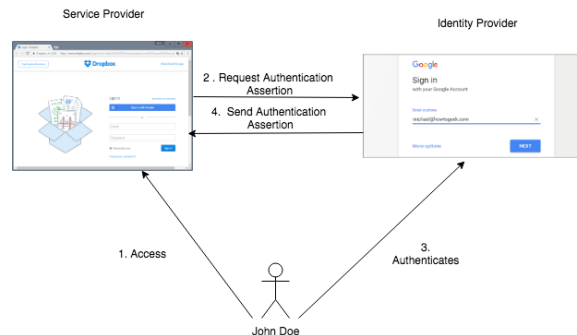
L'esempio mostrato sopra fa parte di uno degli scenari più semplici di servizi di Single Sign On, in quanto abbiamo uno scenario di Single Sign On in cui un utente vuole accedere al servizio dello stesso service provider, che fornisce anche l'identità digitale per accedere a questo servizio. Possiamo, allora, avere anche scenari più complessi di Single Sing On, in cui un'identità fornita da un service provider può essere utilizzata per accedere a servizi, che non sono forniti dallo stesso service provider. Vediamo un esempio di questo scenario:



Immaginiamo, quindi, che l'utente abbia le credenziali del sito EasyJet.com e quello che tipicamente succede, è che oltre il volo, l'utente deve prenotare anche l'hotel e una macchina e quindi idealmente, l'utente dovrebbe autenticarsi nuovamente, con un nuovo set di credenziali, su Hertz.com e su Booking.com. In realtà, in uno scenario di Cross-domain authentication, l'utente si può autenticare con le sue credenziali di EasyJet.com una volta e poi, quando deve prenotare l'hotel e/o la macchina, non dovrà ri-autenticarsi, bensì EasyJet fornirà agli altri due siti un'asserzione del processo di autenticazione effettuato → per riuscire a fare questo, **è necessario stabilire una relazione di fiducia tra chi fornisce l'identità digitale e i servizi digitali, che utilizzano tali identità digitali per concedere agli utenti di utilizzare i propri servizi e le proprie risorse online**. Tipicamente, allora, si stabilisce una **coalizione di service provider** e si stabilisce quali service provider, all'interno della coalizione, possono ricoprire il ruolo di identity provider, ovverosia di fornitori di identità digitali agli utenti dei servizi e di verifica della loro identità digitale → per riuscire a fare ciò, è però necessario, che vi sia un meccanismo che consenta agli identity provider e ai service provider all'interno della coalizione, di scambiarsi informazioni sia sull'identità digitale degli utenti sia sul contesto di autenticazione effettuato dagli identity provider. Per questo motivo, quindi, lo standard utilizzato è **SAML (Security Assertion Markup Language)**, il quale è basato sul linguaggio XML. XML viene utilizzato essenzialmente per due scopi:

1. fornire un formato standard, che consente al service provider di richiedere all'identity provider di autenticare l'utente, che vuole accedere ai servizi;
2. come formato standard, al fine di specificare la risposta che l'identity provider restituisce al service provider su un processo di autenticazione effettuato dall'utente.

Vediamo a questo punto, allora, un esempio di utilizzo di SAML:



Abbiamo un utente, il quale ad esempio si vuole collegare a Dropbox, utilizzando le credenziali di Google. Quindi, quando l'utente tenta di collegarsi a Dropbox, quest'ultimo (in qualità di service provider) genera una SAML Request, in cui chiede a Google di verificare l'identità digitale dell'utente che vuole accedere a Dropbox. L'identity provider, ovverosia Google:

- verifica l'identità dell'utente, se quest'ultimo non è ancora autenticato;
- oppure se l'utente è già autenticato, allora Google rilascia un'asserzione, che dimostra che l'utente è stato autenticato con successo, e tale asserzione la manda a Dropbox → tipicamente, l'asserzione è firmata con la chiave privata dell'identity provider, per evitare che venga modificata e quindi garantirne l'originalità.

A questo punto, Dropbox verifica la firma dell'asserzione attraverso la chiave pubblica dell'identity provider e se la verifica della firma va a buon fine, allora l'utente può collegarsi a Dropbox e accedere alle proprie risorse online → per riuscire a svolgere questo processo, quindi, SAML supporta un linguaggio standard, ovverosia XML, per specificare due cose:

1. la richiesta;
2. le informazioni sul processo di autenticazione, quali gli attributi che sono stati utilizzati per autenticare l'utente.

Quindi, abbiamo due tipologie di **formati standard**, chiamati **Statements**, supportati da SAML ed in particolare, questi due formati standard sono:

1. **Authentication Statement** → il quale viene incluso nella risposta, che l'identity provider invia al service provider, dopo che quest'ultimo ha chiesto di autenticare l'utente;
2. **Attribute Statement** → il quale può essere incluso nella risposta, che l'identity provider invia al service provider, al fine di specificare una lista di attributi associati all'utente che si vuole autenticare.

(Nello scenario di verifica dell'identità digitale non viene utilizzato, ma SAML ha anche un formato standard per specificare i permessi associati ad un utente in relazione ad un determinato servizio).

Quindi, la richiesta che un service provider manda all'identity provider per validare l'identità digitale di un utente, è formata dai seguenti attributi/parametri:

- **identificativo univoco** della richiesta;
- **versione** del **protocollo** SAML;
- **momento** in cui è stata fatta la richiesta;
- **URL** del service provider, che andrà a validare l'asserzione restituita dall'identity provider;
- **<subject>**, per specificare l'**utente** che richiede l'autenticazione;
- **<issuer>**, per identificare il **service provider** che ha effettuato la richiesta;
- **<NameIDPolicy>**, per specificare il livello di sicurezza che deve essere garantito dal protocollo di autenticazione.

La risposta, invece, dell'identity provider ha i seguenti attributi/parametri:

- **identificativo univoco** della risposta;
- **versione** del **protocollo** SAML;
- **momento** in cui è stata generata la risposta;
- **InResponseTo**, che indica l'identificativo della richiesta che è stata fatta;
- **indirizzo** del service provider, che andrà a processare la risposta fornita dall'identity provider;
- **<Status>**, il quale indica se l'autenticazione ha avuto successo oppure no;
- **<Issuer>**, il quale identifica l'identity provider;
- **<Assertion>**, il quale contiene informazioni sul metodo di autenticazione (come ad esempio l'intervallo di tempo fino a quando l'asserzione è valida, oppure l'AudienceRestriction, che specifica che l'asserzione deve essere validata solamente dal service provider che ha richiesto l'autenticazione dell'utente) ed eventualmente sull'identità dell'utente;
- **<Signature>**, il quale contiene la firma digitale dell'identity provider.

Un esempio di protocollo che implementa il contratto di **federazione** è lo **SPID**, il quale è la nostra identità digitale a livello nazionale. Lo SPID, quindi, rispecchia a

pieno il concetto di federazione, in quanto abbiamo una serie di service provider, che possono fornire l'identità digitale (alcuni esempi sono: PostelID, TIM id oppure SpidItalia, che sono sia service provider sia identity provider) e poi abbiamo una serie di servizi a cui possiamo accedere attraverso la nostra identità digitale e per fare questo viene utilizzato SAML. Un altro esempio di protocollo che utilizza SAML è il protocollo **Shibboleth** → Shibboleth consente alle università di condividere risorse e ricerche oltre i confini istituzionali, ovverosia Shibboleth consente a studenti, docenti e personale di accedere alle risorse delle istituzioni partner senza dover creare ID e password locali separati per ciascuna di esse.

---

## OpenID Connect

OpenID Connect è un protocollo di autenticazione, che si basa su un altro protocollo, ovverosia Open Authorization, che è essenzialmente un protocollo di autorizzazione. Open Authorization viene utilizzato in tutti quei contesti, in cui abbiamo un utente che vuole garantire ad un'applicazione Web il permesso di accedere a delle sue risorse, le quali sono ospitate su un Server HTTP → **OAuth**, quindi, viene utilizzato per delegare ad un'applicazione Web di accedere alle proprie risorse ospitate su un Server HTTP. OAuth, però, copre solamente la parte di autorizzazione e quindi non va a verificare, che l'utente che garantisce il permesso, sia effettivamente un utente autenticato e proprietario delle risorse ospitate sul Server HTTP. OpenID Connect, allora, va ad aggiungere la parte di autenticazione.

Per capire tutto il contesto, dobbiamo innanzitutto capire il funzionamento del protocollo OAuth, il quale ha quattro attori principali:

1. **Proprietario della risorsa (Resource Owner)** → entità in grado di concedere l'accesso a una risorsa protetta. Si tratta, quindi, dell'utente che ha le proprie risorse ospitate su un Server HTTP;
2. **Client** → applicazione Web di terze parti, che richiede l'accesso a risorse protette per conto del proprietario della risorsa e con la sua approvazione.
3. **Server della risorsa (Resource Server)** → Server che memorizza le risorse del proprietario della risorsa;
4. **Authorization Server** → Server che rilascia il token di accesso (detto **access token**) al Client dopo aver autenticato il proprietario della risorsa e aver ottenuto la sua autorizzazione → successivamente, l'accesso token può essere presentato al Resource Server (che di fatto è il Server HTTP che ospita le risorse dell'utente) come prova che all'applicazione è stato delegato il permesso per accedere alle risorse dell'utente;

Il flow base di OAuth viene chiamato **Authorization Code Flow** costituito da:

- fase iniziale del protocollo OAuth, in cui l'applicazione Web a cui l'utente garantire il permesso di accedere alle sue risorse (ospitate su un Server), si deve registrare → durante la fase di registrazione, quindi, si deve specificare:
  - il nome dell'applicazione che si vuole utilizzare;
  - un indirizzo, a cui l'utente verrà rediretto nel momento in cui il Resource Owner avrà garantito l'autorizzazione all'applicazione per accedere alle risorse.

Dopodiché, l'Authorization Server restituisce all'applicazione due informazioni:

1. l'**identificativo univoco dell'applicazione**;
2. un **client-secret**, il quale deve essere memorizzato in maniera sicura dall'applicazione.

Una volta terminata questa prima fase di registrazione, l'utente (ovverosia il Resource Owner) può iniziare la richiesta di autorizzazione presso L'Authorization Server. Immaginiamoci, quindi, di avere un utente che vuole utilizzare un'applicazione per accedere al suo conto bancario online (online banking). Il primo passo del flusso di autorizzazione, quindi, è di contattare l'Authorization Server, al fine di ottenere un access code per l'applicazione Web (ovverosia l'online banking), in modo tale da poter accedere alle informazione del conto bancario. In questa prima fase, quindi, viene inviata una POST Request contenente:

- il nome dell'applicazione;
- il redirect URI, che era stato specificato in fase di registrazione;
- il tipo di authorization flow, che voglio che venga supportato.

Dopodiché viene inviata la richiesta e il browser dell'utente viene rediretto all'Authorization Server e vengono specificati dei parametri:

1. il client id;
2. il redirect uri;
3. lo scope, che in questo caso è il bank account;
4. un valore di state, che serve per prevenire reply attack.

A questo punto, l'Authorization Server chiede all'utente di autorizzare l'applicazione di online bank per accedere alle sue risorse ed una volta che l'utente autorizza l'applicazione, viene generato ed inviato all'Authorization Server un authorization

code (il quale è codificato in base 64). Dopodiché, l'utente deve scambiare l'authorization code con un access code e per fare questo, l'utente contatta nuovamente l'Authorization Server e gli passa:

- il codice di autorizzazione;
- il redirect URI;
- il client id;
- il client-secret, che gli era stato dato in fase di registrazione.

A questo punto, l'Authorization Server risponde con due informazioni:

1. l'access token, che è la prova che indica che l'applicazione è stata autorizzata;
2. il refresh token → può essere utilizzato dall'applicazione per ottenere un nuovo access token, senza rieseguire tutti i passaggi visti fino a questo punto, nel momento in cui l'access token precedente non è più valido.

A questo punto, l'access token viene utilizzato dall'applicazione per accedere al Resource Server.

**“Come cambia il flusso, nel momento in cui implementiamo anche il livello di autenticazione del Resource Owner?”** La fase iniziale rimane invariata, ed in particolare il primo cambiamento si ha quando l'applicazione invia un'authorization request all'Authorization Server, oltre a specificare le informazioni del profilo dell'utente a cui si vuole accedere, si deve anche specificare che l'applicazione vuole ottenere un token open-id dall'Authorization Server.

Dopodiché, l'utente autorizza l'applicazione ad accedere alle proprie risorse (esattamente come succedeva prima) e l'Authorization Server restituisce un authorization code all'applicazione e dopodiché, l'applicazione scambia l'authorization code con un access code, andando ad inviare all'Authorization Server l'authorization code ottenuto e l'Authorization Server invia:

- l'access token;
- un identity token → il quale mostra che l'utente è stato autenticato dall'Authorization Server. In questo token è contenuto:
  - l'identificativo dell'utente;
  - il riferimento a chi ha verificato l'identità dell'utente (che nel nostro caso è l'Authorization Server);

- un riferimento all'applicazione che deve verificare la validità del token (nel nostro caso l'online bank);
- il momento in cui l'autorizzazione è avvenuta;
- il periodo di tempo per cui il token è valido.