POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Integrazione di OAuth 2.0 ed OpenID Connect nell'infrastruttura pan-Europea di identità digitale eIDAS

Relatori
Prof. Antonio Lioy
Dr. Ing. Diana Gratiela Berbecaru

Candidato
Gianluca Morabito

Anno accademico 2022-2023

Ai miei genitori
A mia sorella

Sommario

Il principio di identità digitale, ampiamente diffuso, applicato ed a volte controverso, racchiude in sè la possibilità di autenticare un'entità (individuo o organizzazione), dotata di specifici e personali attributi che la descrivono, contro un particolare sistema digitale, al fine di garantire comunicazioni e transazioni (che solitamente si svolgono online) sicure ed al contempo efficienti in un era in cui il trend per l'impiego di sistemi altamente interconnessi, anche a livello globale (ed a volte anche oltre), è in fase esponenzialmente crescente.

Quanto introdotto descrive quindi uno dei concetti chiave e ricopre un ruolo fondamentale nel modo di intendere, approcciarsi e gestire la sicurezza digitale, in cui il concetto di identità digitale si intreccia e si associa a quello di autenticazione digitale, secondo un modello che difficilmente permette e prevede l'esistenza di una senza l'altra e viceversa.

Restringendo il campo d'azione, l'infrastruttura pan-Europea di identità digitale, governata dalla regolamentazione eIDAS (Electronic IDentification, Authentication and trust Services), prevede un framework in grado di abilitare la verifica e l'autenticazione dell'identità attraverso e oltre i confini nazionali dei paesi europei aderenti all'iniziativa.

L'obiettivo, in questo caso, è quello di fornire ai cittadini, alle organizzazioni pubbliche e private ed ai relativi governi, uno strumento per garantire affidabilità e identificazione nei sistemi digitali secondo un approccio integrato tra gli stati membri, in cui ad un cittadino con paese di origine B sia permesso di accedere ad un servizio o ad una risorsa offerti da un fornitore in un paese A, in modo agevole e facilmente attuabile, senza la necessità di effettuare registrazioni ulteriori al nuovo sistema.

I casi d'uso sono molteplici e si estendono dalla possibilità di accedere a servizi offerti online dai governi locali per l'adempimento dei doveri e la fruizione dei diritti dei cittadini, all'integrazione internazionale dei dati personali, anche i più sensibili e privati, quali ad esempio i dati sanitari, naturalmente, solo qualora autorizzati esplicitamente dal cittadino richiedente, passando anche per le transazioni finanziarie o e-commerce che mettono in relazione i governi con i business ed i cittadini.

Dal punto di vista tecnologico, come indicato precedentemente, esistono molteplici approcci per implementare i meccanismi per la gestione e manutenzione delle identità digitali e permettere i relativi flussi per l'autenticazione della stessa. In particolare, uno degli standard attualmente più diffusi, è il SAML (Security Assertion Markup Language), definito nella sua versione 2.0, al giorno d'oggi adottata, nel Marzo 2015 da parte dell'organizzazione OASIS (Organization for the Advancement of Structured Information Standards).

L'approccio si basa in questo caso sull'utilizzo di assertion contenenti le informazioni di richiesta e di risposta per gestire un'autenticazione digitale, comprendendo sistemi complessi di codifica e decodifica dei dati. SAML rappresenta quindi lo standard anche per la gestione dell'identità digitale nell'infrastruttura europea eIDAS, permettendo la gestione di identità federate, secondo i principi chiave di equivalenza tra stati e relativi schemi per l'identità, riconoscimento reciproco tra le nazioni coinvolte e interoperabilità oltre confine.

Tuttavia, si sta registrando oggi, un crescente interesse per l'utilizzo e la standardizzazione di tecnologie alternative rispetto al datato e macchinoso SAML. Si tratta in particolare di OAuth 2.0 e OpenID Connect (OIDC): due protocolli strettamente correlati, il secondo infatti, si pone come livello aggiuntivo rispetto al primo per garantire autenticazione oltre che autorizzazione, considerati sicuramente più moderni nell'apporccio e nella gestione e flessibili.

La forza di OAuth 2.0 e OIDC risiede, infatti, nell'impiego di strumenti più leggeri e rapidi per la gestione dei flussi di autenticazione. Le tecnologie alla base sono le RESTful API e il formato JSON, considerate uno degli approcci favoriti al giorno d'oggi in cui la maggior parte delle interazioni avviene con dispositivi mobili, smart ed IoT.

Questo progetto di tesi mira dunque ad esplorare la fattibilità ed i potenziali vantaggi offerti dall'integrazione di OAuth 2.0 e OIDC nel sistema eIDAS, con un focus rivolto agli aspetti tecnici di implementazione dell'integrazione, per la quale verrà fornita una proposta di soluzione, valutando anche i vantaggi e gli svantaggi rispetto all'approccio attuale basato su SAML.

In particolare, per completare l'integrazione con eIDAS, verrà impiegato un provider di identità, attualmente standard per i sistemi di autenticazione nazionali italiani, SPID (Sistema Pubblico di Identità Digitale), che adotta lo standard OAuth 2.0 con OIDC per la gestione delle sue identità e si presta quindi al caso d'uso descritto dall'integrazione proposta.

Al termine dell'analisi che terrà in considerazione gli aspetti di sicurezza, le performance, la user experience ed eventuali ulteriori implicazioni per il framework europeo, verranno presentate le considerazioni finali e proposte idee per soluzioni future, prima di riportare due guide (developer manual e user manual) che descrivono dettagliatamente i passaggi necessari per portare a termine l'integrazione di OAuth 2.0 con OIDC nel sistema eIDAS secondo le modalità proposte e fornire le indicazioni per testarne il funzionamento.

Indice

Capitolo 1

Introduzione

L'identità digitale rappresenta al giorno d'oggi un tassello fondamentale della società moderna, che la adotta ad ampio raggio, il più delle volte inconsapevolmente, o poco consapevolmente, in un'era dominata dal crescente utilizzo di tecnologie digitali e di internet. Con l'identità digitale si predispone uno scudo a protezione di dati ed informazioni sensibili, comunemente dati personali e riservati, teoricamente invalicabile dall'esterno in assenza del consenso di chi possiede le informazioni stesse.

La gestione e la protezione delle informazioni è quindi diventato un aspetto critico del mondo digitale interconnesso, che richiede delle soluzioni robuste ed adeguate ai numerosi scenari e casi d'uso. Analizzando l'identita digitale è anche necessario trattare l'aspetto dell'identità federata, ad essa strettamente correlata, che si riferisce ad un sistema in grado di abilitare un utente ad autenticarsi ed accedere a più applicazioni e servizi con un unico insieme di credenziali.

Nell'ambito dell'infrastruttura pan-Europea di identità digitale, gli attuali sistemi si basano perlopiù su Security Assertion Markup Language (SAML), per l'appunto considerato l'attuale standard per single sing-on (SSO) e per la gestione di identità digitale ed identità federate. Tuttavia, l'integrazione di nuovi protocolli come OAuth 2.0 con OpenID Connect (OIDC) può rappresentare un nuovo approccio al problema di gestire in modo sicuro ed efficiente le identità digitali.

L'obiettivo della presente tesi è quindi quello di produrre uno studio di fattibilità, con successiva applicazione pratica della soluzione, analizzando anche gli eventuali benefici che potrebbe apportare l'integrazione dei due protocolli OAuth 2.0 e OIDC nell'infrastruttara pan-Europea di identità digitale, confrontando, infine, il nuovo sistema con quello attuale basato su SAML. Il progetto procederà quindi con l'analisi delle differenze tecniche e potenziali miglioramenti e limitazioni di questo nuovo approccio, considerando gli aspetti di sicurezza, efficienza ed usabilità.

1.1 Overview dell'infrastruttura di identità digitale

Per identità digitale si intende la rappresentazione online di un individuo o un'organizzazione, all'interno di un particolare sistema informatico, utilizzata per diversi scopi quali ad esempio transazioni online, accesso a servivi e comunicazioni. Nel mondo di oggi, sempre più digitalizzato, l'identità digitale gioca quindi un ruolo critico per permettere ad individui e organizzazioni di interagire reciprocamente in modo sicuro ed efficiente. Di conseguenza, la gestione delle identità digitali, è diventata uno dei maggiori aspetti di interesse per business, entità governative e altre organizzazioni online.

Con infrastruttura di identità digitale si intende lo strato di sistemi e tecnologie adottate per gestire e rendere sicure le identità digitali. Questo tipo di infrastruttura risulta essenziale in funzione di una società sempre più digitalizzata. Uno dei concetti chiave è quello di identità federata, che si riferisce ad un meccanismo in cui più di un'organizzazione accetta l'utilizzo di

un'unica identità digitale (solitamente riferita ad un individuo) per garantire l'accesso a servizi di organizzazioni differenti, riducendo quindi la necessità, da parte di fornitori e fruitori, di gestire più identità digitali per rappresentare un'unica entità o individuo.

Lo sviluppo di un'infrastruttura di identità digitale è stato guidato dal crescente bisogno di soluzioni sicure, comode ed interoperabili per la gestione dell'identità digitale, permettendo un'evoluzione esponenziale negli ultimi anni, che ha rimarcato il miglioramento, o a volte, la necessità di miglioramento, delle tecnologie e degli aspetti di sicurezza sottostanti.

Sono quindi state sviluppate numerose soluzioni per l'utilizzo dell'identità digitale, con lo scopo di permettere una completa e sicura gestione della stessa, con riferimento quindi a protocolli come SAML e OAuth 2.0 con OIDC. Ognuna delle soluzioni proposte è dotata di caratteristiche uniche, benefici e limitazioni e risulta quindi necessario valutarne i componenti chiave per produrre decisioni consapevoli riguardo l'adozione delle stesse nelle relative infrastrutture di identità digitale.

1.1.1 Autenticazione elettronica

L'autenticazione elettronica, soventemente correlata al principio di identità digitale, indica il processo da attuare per stabilire la fiducia tra uno user autenticato digitalmente e un sistema digitale. Ci sono diversi metodi per permettere l'autenticazione elettronica che variano dall'utilizzo di semplici coppie di username e password sino a più sofisticati e sicuri meccanismi che implicano l'utilizzo di multi factor authentication (MFA). In particolare, i fattori di autenticazione in gioco possono essere relativi a una particolare conoscenza (knowledge factor), a un possedimento (possession factor) o a caratteristiche biometriche (biometric factor). I token, ad esempio, sono degli strumenti forniti da un gestore di identià ad un utilizzatore che li possiede, ne dispone e li controlla offrendo la possibilità di verificare l'autenticità dell'identità di chi richiede di essere riconosciuto (tipicamente uno user).

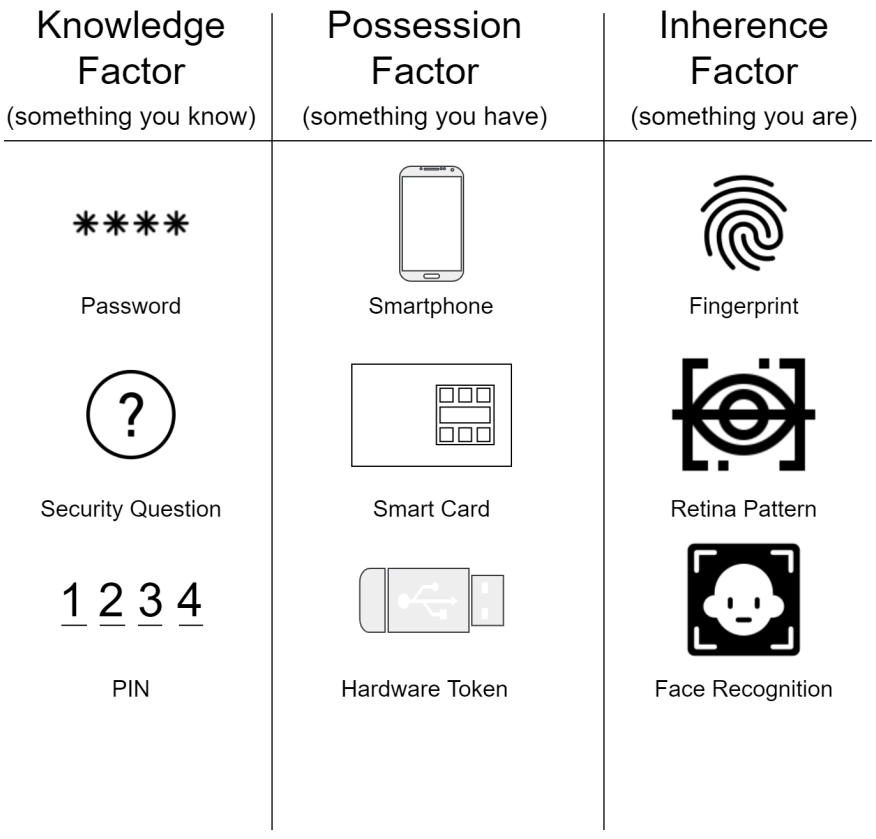

Figura 1.1. Tipologie di fattori di autenticazione (source: Rublon).

Seguendo il modello per l'autenticazione elettronica generica sviluppato dal NIST, un Credential Service Provider (CSP), è l'entità a cui viene richiesto di verificare, seguendo degli step preliminari, l'effettiva corrispondenza dell'identità di chi richiede di essere autenticato in fase di registrazione al sistema, prima di procedere con qualsiasi tipo transazione online. Dopo questo primo step identificativo, il richiedente, riceve infatti un sistema per l'autenticazione, quale ad esempio un token, e/o delle credenziali, per esempio username e password. Il richiedente può quindi successivamente identificarsi presso il sistema presentando il token ed adottando lo specifico protocollo di autenticazione richiesto. Questo meccanismo viene chiamato Proof of Possession.

1.2 Importanza dell'autenticazione dell'identità digitale

Nell'odierno mondo digitale, l'autenticazione dell'identità digitale risulta cruciale sia per gli individui che per le organizzazioni. Essa permette infatti l'accesso sicuro alle risorse e ai servizi esposti online, proteggendo le informazioni sensibili ed assicurando che le transazioni vengano condotte in modo sicuro ed affidabile. Con l'aumento dell'uso di internet e la crescita dell'e-commerce, l'autenticazione dell'identità digitale, è diventata una componente essenziale nella nostra routine giornaliera, in particolare, per i business e le organizzazioni che devono costantemente assicurare privacy e sicurezza per le informazioni ritenute sensibili, ad esempio le transazioni finanziarie o i dati sanitari, e quindi adottare i requisiti forniti dalle varie regolazioni in materia, proteggendo, non per ultima, anche la propria reputazione.

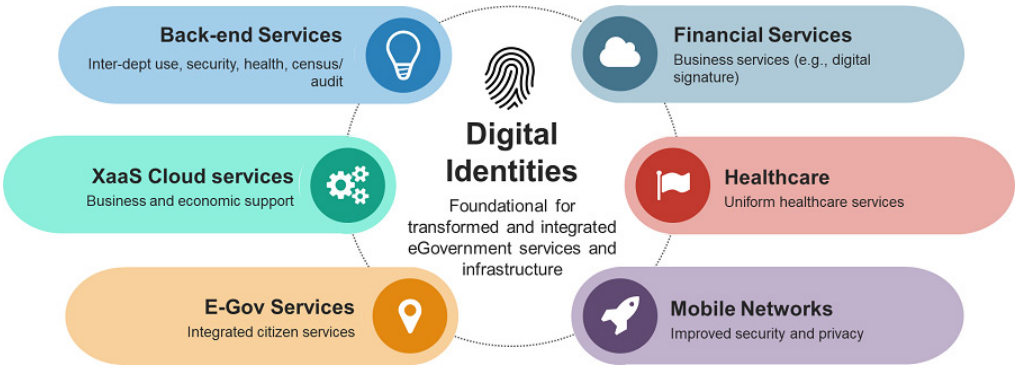

Figura 1.2. Possibili use case per l'identita' digitale (source: Huawei).

Nel panorama globale fortemente interconnesso di oggi, l'autenticazione dell'identità digitale è diventata, inevitabilmente, progressivamente, anche più complicata e delicata. Si richiede quindi l'utilizzo di sistemi di autenticazione più resistenti, in grado di verificare effettivamente l'identità degli user e al tempo stesso mantere alto il livello di privacy delle informazioni da loro condivise.

L'autenticazione dell'identità digitale gioca quindi un ruolo strategico anche nell'abilitare un accesso sicuro ai servizi e alle applicazioni online, come social media, email, banche e e-commerce. Con la crescita dei servizi basati su cloud e il mobile computing, poi, gli user accedono a queste risorse e servizi da una varietà di dispostitivi, rendendo necessario l'impiego di metodi di autenticazione sicuri e comodi in tal senso, pronti ad affrontare le sfide del preponderante trend che spinge verso l'utilizzo in completa mobilità.

Alcuni metodi di autenticazione, basati ad esempio sul semplice impiego di username e password, si sono dimostrati infatti inadeguati nel proteggere da furto d'identità, hacking e altre minacce di sicurezza, altri, invece, hanno fatto emergere criticità nella gestione se adottati in dispositivi nuovi ben lontani dall'era del personal computer in postazione fissa. Sono quindi queste alcune delle motivazioni per cui meccanismi più avanzati e moderni stanno venendo adottati, divenendo sempre più popolari e diffusi.

In relazione a quanto detto sopra, integrazioni come quella di OAuth 2.0 con OIDC nell'infrastruttura pan-Europea di identità digitale, possono giocare un ruolo chiave per affrontare le sfide sopracitate e permettere un adeguamento dei sistemi alle necessità di individui e organizzazioni in continuo divenire e costante frenetico evolvere.

1.3 Scopo del progetto di tesi

Lo scopo primario del progetto di tesi è quello di valutare la fattibilità e gli eventuali benefici derivanti dall'integrazione di protocolli per l'autenticazione sicura come OAuth 2.0 e OIDC nell'infrastruttura pan-Europea di identità digitale, che attualmente si basa su standard come SAML. L'obiettivo finale è, infatti, quello di ridurre le limitazioni ed i lati negativi dell'attuale sistema fornendo una soluzione più flessibile, scalabile, sicura e moderna per l'autorizzazione e l'autenticazione relativa all'identità digitale.

Attraverso gli approfondimenti ed una proposta di implementazione tecnica dell'integrazione menzionata, questa tesi, si pone come traguardo quello di contribuire al complesso avanzamento tecnologico dell'infrastruttura di identità digitale in Europa, permettendo infine il raggiungimento e l'adozione di un approccio unificato, interoperabile ed a misura di utente, per l'accesso ai servizi online, alle applicazioni e alle varie risorse offerte dai governi e dalle organizzazioni private. Peridipiù, questo approccio modernizzante, può anche essere rilevante per la crescita dell'economia digitale, l'e-government e gli e-services, traendo la sua forza dall'abilitazione a transazioni ed interazioni sicure e senza intoppi attraverso i confini dei paesi aderenti all'iniziativa.

In conclusione, lo scopo del progetto di tesi è quello di esplorare i potenziali vantaggi dell'integrazione di OAuth 2.0 e OIDC nell'infrastruttura pan-Europea di identità digitale e di considerare la fattibilità e l'impatto di questa integrazione sull'attuale sistema di autorizzazione e autenticazione dell'identità digitale, valutando e proponendo anche eventuali ulteriori soluzioni per scenari futuri.

Capitolo 2

Revisione della letteratura

2.1 Background su SAML

SAML è uno standard open basato su XML utilizzato per lo scambio di dati per l'autorizzazione e l'autenticazione tra entità, solitamente identificate con Identity Provider (IdP) e Service Provider (SP). Fu sviluppato inizialmente dal consorzio per gli standard di sicurezza OASIS, agli inizi degli anni duemila, e da allora viene adottato in molti sistemi che richiedono questo tipo di funzionalità.

Uno use case chiave per l'utilizzo di SAML è il web browser SSO, in cui uno user ha la possibilità di effettuare il login una volta e accedere poi a una moltitudine di servizi senza la necessità di inserire nuovamente le sue credenziali. Nello scenario descritto, l'IdP è identificabile nell'entità che autentica lo user e genera l'assertion di risposta di tipo SAML, contente le informazioni sull'identità dello user ed eventuali attributi autorizzati dallo stesso; il SP, invece, è l'entità che invia l'iniziale assertion SAML di richiesta e, a seguito dell'autenticazione ricevuta dall'IdP, utilizza le informazioni ottenute per garantire all'utente l'accesso ai propri servizi.

2.1.1 Flusso per SAML web browser SSO

Il tipico flusso per lo scenario di SAML web browser SSO può essere descritto come di seguito:

1. Lo user richiede di accedere a una risorsa protetta sul website del SP.
2. Il SP crea quindi una richiesta (assertion) SAML e redireziona lo user verso l'IdP per l'autenticazione.
3. Lo user fornisce le credenziali all'IdP, che infine lo autentica.
4. L'IdP genera una risposta (assertion) SAML inviandola poi al SP attraverso il browser dello user.
5. Il SP utilizza quindi le informazioni nell'assertion SAML ricvuta per determinare se è possibile garantire allo user l'accesso ai propri servizi.

Il SSO con SAML è realizzabile anche senza includere il browser dello user, utilizzando ciò che è conosciuto come SAML artifact binding[1], ma è una caso d'uso meno comune.

[1] Un artifact, che è un riferimento all'assertion SAML, viene scambiato direttamente tra IdP e SP.

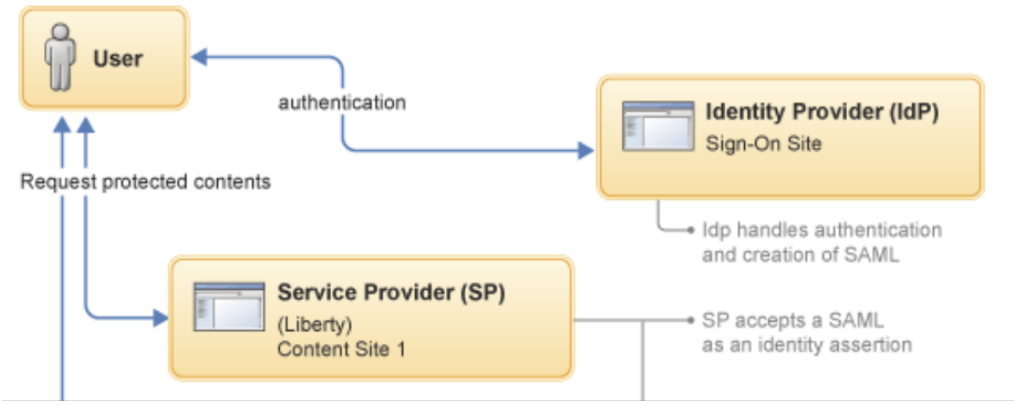

Figura 2.1. SAML 2.0 Web Browser Single-Sign-On (source: IBM).

2.1.2 Pro e contro di SAML

SAML è stato ampiamente adottato negli anni e risulta quindi ben supportato da molti prodotti per la gestione di identità e la sicurezza. Alcuni dei vantaggi sono:

- Alti livelli di sicurezza: SAML utilizza firme digitali ed encryption per proteggere l'integrità e la confidenzialità delle assertion, rendendolo più sicuro di tecnologie alternative come basic authentication o cookie.
- Identità federate: SAML permette la creazione di relazioni di fiducia tra IdPs e SPs, permettendo agli user di accedere a servizi di più organizzazioni usando uno solo set di credenziali.
- Flessibilità: SAML supporta una moltitudine di meccanismi di autenticazione, da username/password a smart card e fattori biometrici, rendendolo una tecnologia versative per molti casi d'uso.

Tuttavia, SAML, presenta anche degli svantaggi, quali ad esempio:

- Complessità: SAML può essere difficile da implementare e configurare, soprattutto per organizzazioni con risorse limitate nella gestione dell'identità e della sicurezza.
- Problemi di interoperabilità: anche se SAML è uno standard ampiamente adottato, ci possono essere sottili differenze nell'effettiva implementazione e gestione di SAML, che possono causare problemi di interoperabilità tra IdPs e SPs.
- User experience più lenta: SAML web browser SSO, per esempio, può presentare dei rallentamenti dal punto di vista della user experience, in quanto lo user viene direzionato verso l'IdP per l'autenticazione e poi nuovamente verso il SP per l'accesso alle risorse protette.
- Oneroso in termini di risorse computazionali: SAML è una tecnologia che può essere considerata "vecchia" in questo mondo digitale rapidamente in evoluzione, progettata quanto la maggior parte delle interazioni digitali avvenivano attraverso personal pc e non smartphone o dispositivi IOT, come avviene diffusamente oggi.

2.2 OAuth 2.0 e OIDC overview

2.2.1 OAuth 2.0

OAuth (Open Authorization) 2.0 e un protocollo open standard per la delegazione dell'accesso, che permette ad applicazioni di terze parti di accedere ai dati utenti salvati in un altro website

o applicazione senza la necessità che l'utente riveli le proprie credenziali di login. Il protocollo si fonda su diversi componenti chiave, tra cui il resource owner, la client application, l'authorization server e il resource server.

Il resource owner è l'individuo proprietario dei dati, come ad esempio le informazioni del profilo a cui l'applicazione client vuole accedere. L'applicazione client è l'applicazione di terze parti che richiede i permessi per accedere ai dati dello user. L'authorization server è l'entità responsabile per approvare o negare l'accesso ai dati del resource owner e che emette i token alla client application. Il resource server infine possiede i dati del resource owner.

OAuth 2.0 definisce diversi tipi di grant, che indicano il modo in cui la client application ottiene accesso ai dati del resource owner. I grant più comuni sono authorization code, client credentials (utilizzato per applicazioni non interattive consigliato per client fidati), implicit (in cui il token viene restituito immediatamente senza aggiuntivi step che prevedono lo scambio di un authorization code), password grant (iin cui iil token è ottenuto in cambio delle credenziali dell'utente, in questo caso il client deve però collezionare anche le credenziali user, interferendo con il reale scopo del meccanismo di delega). Tuttavia, il tipo implicit e password sono considerati legacy e non più utilizzati nè consigliati oggigiorno.

Un esempio di flusso con OAuth 2.0 (caso authorization code grant) è il seguente, figura 2.2:

- Step preliminari: prima di procedere con il flusso definito da OAuth 2.0, è necessario per il client (SP) e l'authorization server (IdP), di concordare i termini di una sorta di partenership, in cui le due entità si riconosco e si scambiano i metadati (informazioni circa gli endpoint coinvolti, tra cui il call-back endpoint del SP, l'authorization, il token ed eventualmente, in caso di OIDC, lo userinfo endpoint dell'IdP, gli algoritmi per la firma digitale e l'encryption e le relative chiavi publiche per la comunicazione sicura).

1. La client application reindirizza lo user all'authorization endpoint dell'authorization server.
2. Lo user effettua il login sull'authorization server a autorizza l'accesso ai propri dati.
3. L'authorization server invia quindi un authorization code alla client application utilizzando il call-back endpoint.
4. La client application scambia l'authorization code per un access token inviandolo al token endpoint.
5. L'authorization server emette quindi un access token per il client.
6. La client application utilizza l'access token per ottenere i dati del resource owner dal resource server.

Nello scenario descritto, la client application ottiene un access token per le risorse ma non può a sua volta autenticare lo user, infatti l'access token permette l'accesso alle risorse del resource owner ma non prevede ulteriori informazioni sull'identità dello user.

Uno dei vantaggi chiave di OAuth 2.0 è che permette un modo sicuro e semplice per gli user di accedere ai propri dati, senza la necessità di condividere le proprie credenziali di login. Questo permette infatti un modello di autorizzazione più flessibile e scalabile, dato che può essere utilizzato in una varietà di casi d'uso, tra cui applicazioni web, mobile e basate su API e che adotta standard come JWT e HTTPS in ogni step del flusso.

Inoltre, c'è un ulteriore modo per ottenere un access token. Un refresh token è una tipologia particolare di token utilizzato per ottenere un nuovo access token. Quando un access token scade, la client application deve ottenerne uno nuovo per poter continuare ad accedere ai dati protetti relativi al resource owner. Anzichè chiedere nuovamente allo user di effettuare il login a garantire gli accessi, è possibile richiedere in precedenza un refresh token ed utilizzare poi questo (con tempi di scadenza più lunghi) per ottenere un nuovo access token.

Il processo funziona come segue:

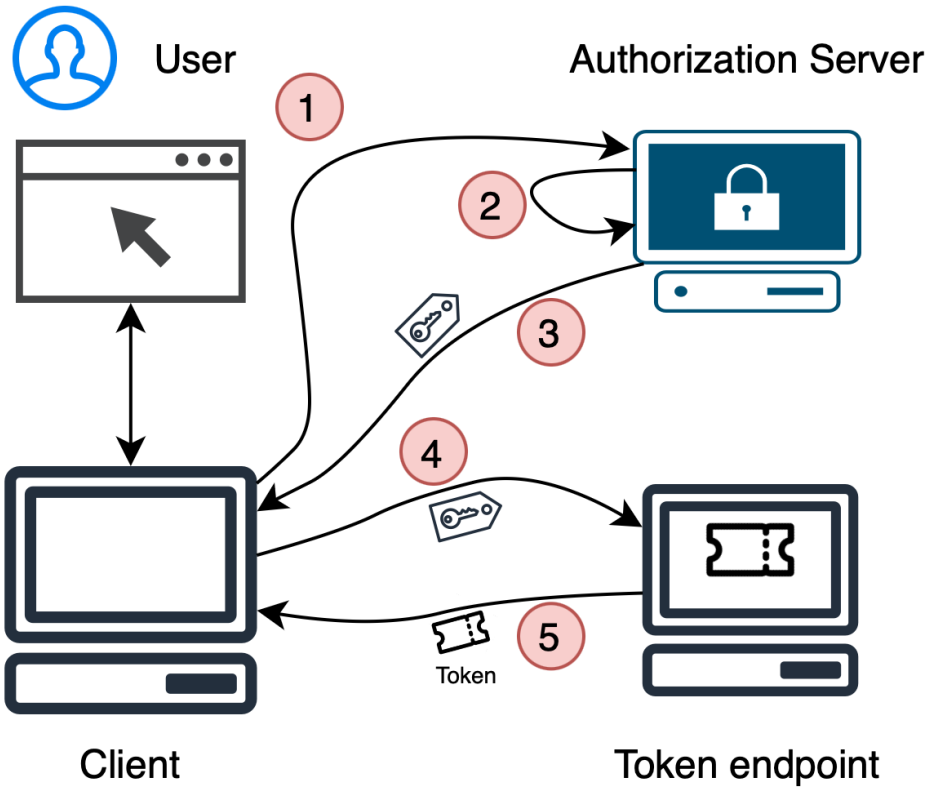

Figura 2.2. Flusso OAuth 2.0 che utlizza authorization code come grant type.

1. La client application invia una richiesta al toke endpoibt dell'authorization server insieme al refresh token ottenuto precedentemente con l'access token (se il meccanismo è stato configurato)
2. L'authorizationo server verifica il refresh token e, se ritenuto valido, emette un nuovo access token.
3. La client application riceve il nuovo access token e lo utilizza per accedere infine nuovamente alle risorse protette.

Il refresh token permette quindi un user experience più fluida, dato che lo user non deve effettuare il login e garantire l'accesso ogni volta che un access token scade. Il meccanismo aumenta inoltre la sicurezza del sistema, permettendo di ridurre il tempo di validità di un access token e il numero di volte in cui uno user deve garantire l'accesso.

Non appare tuttavia chiara da questo processo la correlazione tra risorse e user, questo è il principale motivo per cui è stato introdotto OpenID Connect.

2.2.2 OIDC (OpenID Connect)

OpenID Connect è un protocollo sviluppato come layer aggiuntivo su OAuth 2.0 che aggiunge funzionalità di autenticazione. Infatti, OAuth 2.0 si concentra esclusivamente su autorizzazione, garantendo agli user la possibilità di consentire l'accesso ai propri dati, mentre, OIDC, aggiunge la possibilità di autenticare uno user e verificare la propria identità.

In OIDC, l'authorization server funge più propriamente da IdP, nel senso che contiene anche le informazioni sull'identità dell'utente e può di conseguenza fornirle alla client application nella

forma di un ID token. L'ID token è un particolare JSON Web Token (JWT) che contiene informazioni sull'dientità dello user, come ad esempio il nome, l'indirizzo email, lo stato di autenticazione, ed è inviato alla client application dal token endpoint insieme all'access token precedentemente descritto.

Uno dei principali vantaggi nell'utilizzo di OIDC è che permette un modo sicuro e standardizzato affinchè le applicazioni possano anche autenticare gli user e accedere alle loro informazioni di identità. Inoltre, permette una user experience più lineare, in quanto la client application può avere informazioni anche circa lo stato di autenticazione dello user.

In conclusione, OAuth 2.0 permette un modo flessibile e sicuro per delegare l'accesso ai dati utente, mentre OIDC estende le funzionalità con l'abilità di autenticare anche l'utente. La coppia di protocolli è abbondantemene impiegata in applicazioni web e modbile, soprattutto in scenari che prevedono l'utilizzo di REST api e JSON, descrivendo un modo sicuro ed efficace, nonchè efficiente, per gestire l'autorizzazione e l'autenticazione in un sistema digitale.

2.3 Confronto tra OAuth 2.0 con OIDC e SAML

Sia OAuth 2.0 con OIDC che SAML sono standard di tipo per autorizzazione ad autenticazione, ma differiscono in diversi aspetti che lo rendono una scelta più appropriata dell'altra in funzione di derminate circostanze.

OAuth 2.0 con OIDC ha un carattere che pone sicuramente al centro l'utente, permettendo un accesso sicuro alle risorse protette. Ideale per applicazioni web e dispositivi mobile, permette l'accesso ai dati utente con la possibilità di autenticare anche lo user, con il livello dato da OIDC.

SAML, invece, è un protocollo per lo scambio di informazioni di autorizzazione ed autenticazione tra due parti, SP e IdP, costituendo una soluzione robusta ma più complicata da realizzare, sia in temini di implementazione che di gestione che di infrastruttura e protocolli.

La principale differenza tra i due è che SAML supporta nativamente il SSO e altri meccanismi di autorizzazione, mentre OAuth 2.0 si focalizza sull'autorizzazione delegata, non prevedendo un vero standard per il SSO. Da un punto di vista tecnico SAML, definisce il formato delle assertion, utlizzando meccanismi completssi di crittografia e una dimensione non ridotta dei messaggi scambiati tra le parti. Dall'altro lato, OAuth 2.0, si basa perlopiù su HTTPS e non definisce in maniera univoca il formato delle richieste/risposte.

La vera forza di OAuth 2.0 risiede nella facilità d'uso e flessibilità: il protocollo, può infatti essere facilemente implementato ed eseguito da dispositivi smart (come ad esempio smartTV) e mobili, da applicazioni web e altri sistemi moderni, rimanendo sempre leggero e integrabile grazie alle numerose librerie disponibili per adattarlo alle diverse tipologie di processi adottati dai SPs. SAML, al contrario, non è stato concepito per un utilizzo con le applicazioni più moderne, rendendolo più difficile da adattare e plasmare in sistemi più recenti.

Al giorno d'oggi SAML è utilizzato perlopiù per SSO in applicazioni gestite dai governi e di tipo enterprise, dove il linguaggio XML è diffusamente adottato nei sistemi di elaborazione back-end. Molti schemi di identificazione forniti dai governi per i cittadini sono basati su SAML.

OAuth 2.0, d'altro canto, è abbondantemente adottato in applicazioni di tipo consumer (ma anche enterprise), per autorizzazione ed autenticazione (con OIDC), principalmente per la spiccata abilità di garantire autorizzazioni per RESTful api, dove l'utilizzo di un access token rende il sistema più semplice e flessibile.

Vantaggi di OAuth 2.0 con OIDC:

- OAuth 2.0 con OIDC è stato relizzato per un approccio che pone al centro l'utente, fornendo allo user strumenti per il controllo sui suoi dati e permettendogli di garantire e revocare l'accesso alle sue risorse.
- Simple and lightweight: OAuth 2.0 with OIDC is simple and easy to implement, making it ideal for smaller projects and mobile devices.

- Secure: OAuth 2.0 with OIDC provides a secure way to access protected resources, using encryption and secure tokens to protect user data.
- Flexibility: OAuth 2.0 with OIDC can be easily integrated into existing systems, making it a flexible solution for a wide range of projects.
- Semplice e leggero: OAuth 2.0 con OIDC è semplice e relativamente facile a livello implementativo, rendendolo la soluzione ideale per progetti più ridotti e che prevedono l'impiego di dispositivi mobili.
- Sicuro: i due standard forniscono un metodo sicuro per l'accesso alle risorse protette, utilizzando encryption (JWE) e token sicuri per proteggere i dati utente.
- Flessibilità:

Limitazioni di OAuth 2.0 con OIDC:

- Informazioni limitate: OAuth 2.0 con OIDC fornisce informazion limitate sull'identità di uno user, in funzione della configurazione adottata, rendendolo non adatto per applicazioni che richiedono un'informazione più robusta circa l'identità.

Vantaggi di SAML:

- Informazione robusta sull'identità: SAML fornisce una soluzione più robusta per lo scambio di informazioni in relazione all'identità dello user, rendendolo più adatto per applicazioni che necessitano informazioni più dettagliate sullo user.
- Ampiamente supportato: SAML risulta ampiamente supportato, rendendolo compatibile con una moltitudine di sistemi e serivizi

Limitazioni di SAML:

- Complessità: SAML è più complesso di OAuth 2.0 con OIDC sotto diversi punti di vista, rendendolo piò complicato da implementare e gestire.
- Infrastruttura: SAML richiede un'infrastruttura più ampia per essere implementato, sia per l'IdP che per il SP.
- Performance. SAML può avere degli aspetti negativi sulle performance dati dalla mole di informazioni scambiate tra le controparti. Peridipiù, non supporta il meccanismo di lunghe sessioni come quello previsto dall'utilizzo di refresh token con OAuth 2.0.

2.4 JSON

JSON (JavaScript Object Notation) è un formato leggero per lo scambio di dati ampiamente adottato in applicazioni web e mobile moderne. Si basa su contenuto testuale che risulta facilmente leggibile, scrivibile e comprensibile dagli esseri umani e, al tempo stesso, facile da analizzare e generare da un computer. Il supporto per il JSON si estende alla maggior parte dei linguaggi di programmazione, nativamente o mediante l'impiego di popolari librerie, che hanno permesso di diventare lo standard de facto per lo scambio di informazioni sul web.

La notazione JSON si basa su collezioni di coppie chiave-valore, in cui le chiavi sono rappresentate sempre da stringhe mentre i valori possono assumere più forme, compatibilmente ai tipi di dato supportati, come stringhe, numeri, booleani, array o anche oggetti JSON stessi nel caso di nested JSON. Gli oggetti JSON sono racchiusi sempre all'interno di parentesi graffe, mentre gli array utilizzano le parentesi quadre. Esistono molteplici standard che si basano sul formato JSON, tra cui JWK, JWS, JWE, JWT.

JWK (JSON Web Key) è uno standard per la rappresentazione di chiavi crittografiche attraverso JSON. Solitamente viene utilizzato in combinazione con JWT per memorizzare e scambiare in modo sicuro le chiavi tra sistemi differenti. Può essere utilizzato per rappresentare chiavi simmetriche, come shared secret, o coppie di chiavi asimmetriche , come chiave pubblica e chiave privata. Le chiavi presenti all'interno dei JWK possono essere utilizzare per firmare e criptare i JWT o per verificare la firma o decriptarne i relativi payload.

JWS (JSON Web Signature) è utilizzato come standard per la creazione e la verifica di firme digitali basata sul formato JSON, imipiegato comunemente per garantire integrità (firma applicata sul digest) e autenticità (firma con chiava asimmetrica privata) dei dati. Un JWS è composto da due parti: un header ed un payload. Il primo contiene le informazioni tecniche relative alla firma, come ad esempio l'algoritmo impiegato e il riferimento della chiave usata, key ID; il payload, invece, contiene propriamente i dati da trasferire. I due campi vengono codificati in base64 e uniti in un'unica stringa, seperati da un ".". Sulla stringa appena prodotta viene quindi applicato un algoritmo di hashin per calcolarne il digest e successivamente, su questo, viene applicata la firma con la chiave privata. Il JWS si compone quindi, infine, di tre campi separati da ".", header, payload, firma ottenuta, anch'essa codificata in base64.

JWE (JSON Web Encryption) è lo standard per cifrare i dati in trasferimento con formato JSON. Viene impiegato infatti per proteggere informazioni sensibili durante le trasmissioni in internet per garantire confidenzialità dei dati. Un JWE si compone di cinque parti, codificate in base64 e separate da un ".": un campop per l'header che contiene le informazioni sull'algoritmo utilizzato per la cifratura dei dati e il riferimento, key ID, della chiave pubblica utilizzata per cifrare la chiave effimera (che garantisce la perfect-forward-secrecy), encrypted key, utilizzata per criptare realmente i dati e presente al campo due del JWE; un campo IV per condividere l'initialization vector usato dall'algoritemo di cifratura, il campo dedicato al cyphertext da decifrare ed infine un campo designato per un tag di autenticazioned che permette di verificare l'integrità del cyphertext e dell'header.

Infine, JWT (JSON Web Token) che solitamente assume la forma di un JWS o di un JWE, o una combianzione dei due con un JWS inserito nel campo cyphertext del JWE (scenario più sicuro), costituisce uno standard open basato su JSON per creare token di accesso sicuri e verificabili, solitamente per l'autenticazione e l'autorizzazione associata a servizi/risorse online e relative API.

2.5 RESTful API

REST (Representational State Transfer) rappresenta una categoria di architettura del software per costruire sistemi distribuiti sul web, costituendo il design dominante per l'utilizzo di API (Application Programming Interface).

Le API di tipo RESTful sono designate per essere semplici, facilmente scalabili e stateless, utilizzano il protocollo HTTP(Hypertext Transfer Protocol) per le comunicazioni adottandone i relativi metodi (GET, POST, PUT, DELETE, ecc...) per effettuare operazioni sulle risorse messe a disposizione.

Uno dei principi chiave per questo genere di API, ed in generale per il paradigma REST, è il design orientato alla gestione di risorse. Le RESTful API, infatti, espondono le risorse sotto forma di URI (Uniform Resource Identitifers) a cui è possibile accedere usanto i metodi dello standard HTTP precedentemente menzionati. Ogni risorsa esposta rappresenta quindi un unità logica di dato, ad esempio un customer, un prodotto, una transazione, un servizio.

Inoltre, una caratteristica chiave, risiede nella proprietà di connessioni stateless tra due entità che utilizzano le API (solitamente un client che invia ed un server che riceve e viceversa): le API infatti, non mantengono alcuna informazione sul context del client chiamante o sullo stato del server chiamato. Tutte le informazioni necessarie sono sempre contenute all'interno della singola richiesta (API), comprese ad esempio le credenziali di autenticazione, permettendo alle RESTful API di essere altamente scalabili e resilienti, senza la necessità di eseguire logiche basate sullo stato del server.

Le RESTful API sono sviluppate secondo un'achitettura a strati, ognuno con il proprio set specifico di funzionalità: livello client, responsabile di mandare le richiesta alle API e ricevere risposte; livello API designato all'elaborazione delle richiesta e delle relative risposte (spesso gestito da un API gateway); livello di dato in cui le informazioni vengono salvata ed estratte in base alle necessità del momento.

Per rendere riscuro questo tipo di API esistono, infine, vari meccanismi tra cui cifratura, autenticazione ed autorizzazione basata su SSL/TLS oppure mediante l'impiego di JWT, JWK, JWE, JWS,spesso utilizzati in combinazione tra loro.

2.6 La European digital identity regulation (eIDAS)

La European Electronic identity and Signature Regulation (eIDAS) è una regolamentazione promossa dall'Unione Europea (UE) nel 2014 al fine di armonizzare il complesso per la gestione della firma digitale, pagamenti digitali ed identità digitale in tutta Europa. Può essere considerata come l'elemento chiave per la gestione dell'identità e l'accesso ai servizi nell'infrastruttura pan-Europea di identità digitale, giocando un ruolo chiave nell'utilizzo di transazioni elettroniche sicure e affidabili attraverso i confini internazionali.

Figura 2.3. Servizi offerti dal framework eIDAS (source: ENISA).

eIDAS prevede un framework legale per le interazioni e le firme digitali, assicurando che tutte le forme di transazione online vengano riconisciute e accettate in modo inequivocabile in tutti i paesi dell'UE. La regolazione è stata realizzata per aumentare il grado di confidenza e sicurezza nelle transazioni elettroniche e promuovere quindi le attività di scambio e commercio oltre confine in Europa. Ulteriore obiettivo è infine quello di proteggere i diritti individuali e la relatività libertà in un mondo digitale, promuovendo l'utilizzo di identità digitali sicure e affidabili.

Si basa su tre principi fondamentali: equivalenza, riconoscimento reciproco e principio di interoperabilità. Il principio di equivalenza significa che le firme elettroniche, i sigilli elettronici e l'identificazione elettronica devono essere trattati come equivalenti ai metodi adottati tradizionalemente dai vari paesi nell'UE. Il principio del riconoscimento reciproco garantisce che le firme elettroniche, i sigilli e il metodo di identificazione emessi in uno Stato membro dell'UE (Member State MS) siano riconosciuti ugualmente in tutti gli altri stati membri. Il principio di interoperabilità infine garantisce che le firme elettroniche, i sigilli e i metodi di identificazione siano interoperabili tra gli Stati membri dell'UE.

eIDAS fornisce diversi livelli di garanzia per le firme elettroniche e l'identificazione elettronica, che vanno da basso ad alto, a seconda del livello di sicurezza richiesto per un particolare tipo di transazione. Questo aiuta a garantire che gli utenti adottino il giusto livello di sicurezza per le loro particolari esigenze e che le transazioni elettroniche possano essere condotte con fiducia.

Per quanto riguarda l'identificazione elettronica, il quadro normativo eIDAS ha messo in atto un protocollo comune da seguire basato sul protocollo SAML al fine di scambiare assertion tra i diversi schemi di eID (identità elettronica) registrati degli stati membri per garantire un'interoperabilità tecnica e semantica.

Il formato del messaggio SAML e i relativi profili di attributi che sono stati adottati, sono stati mantenuti uguali nei vari profili eID di eIDAS. Tuttavia, questa scelta di implementazione non influisce direttamente sugli utenti, poiché gli IdP rimangono liberi di adottare un altro tipo di protocollo per gestire i propri sistemi di identificazione (ad esempio, OAuth 2.0, OIDC). Il protocollo eIDAS, infatti, viene utilizzato solo per le assertion di autenticazione e le informazioni sull'identità scambiate tra i nodi eIDAS e oltre i confini.

La scelta del protocollo SAML è stata fatta in un periodo in cui la maggior parte delle interazioni digitali avveniva attraverso l'utilizzo di pc desktop, ma negli ultimi anni gli smartphone sono diventati il mezzo preferito di interazione digitale per la maggior parte dei cittadini dell'UE. Nel 2018, l'86% delle persone con un'età compresa tra i 16 e i 74 anni, che ha effettuato un accesso a Internet in un periodo di 3 mesi, lo ha fatto tramite uno smartphone.

Utilizzando SAML, quindi, il flusso di autenticazione messo in atto utilizzando un'applicazione per smartphone, ad esempio, potrebbe rischiare di essere interrotto durante il complesso processo, sicuramente molto più intricato e lento di quanto lo sarebbe utilizzando altre soluzioni più leggere e moderne.

2.6.1 Architettura di interoperabilità eIDAS

Definizioni:

- MS: stato membro dell'UE in cui è attiva la regolamentazione eIDAS.
- Person: può essere naturale o legale.
- Sending MS: lo stato membro a cui è richiesto di fornire gli ID data autenticati al receiving MS.
- Receiving MS: lo stato membro a cui un'entità di terze parti invia la richiesta per l'autenticazione di un individuo.
- eIDAS-Node: nodo operativo coinvolto nel flusso transfrontaliero di autenticazione di una persona. Può essere un eIDAS-Connector, quello che richiede l'autenticazione, o un eIDAS-Service, quello che fornisce l'autenticazione. In particolare, l'eIDAS-Service può essere di due tipi: eIDAS-Proxy-Service, gestito dal sending MS e che fornisce direttamente i dati identificativi, o eIDAS-Middleware-Service, gestito dal receiving MS per conto dello sending MS e che fornisce gli ID data.
- Proxy based scheme: lo schema utilizzato per l'autenticazione tramite eIDAS-Proxy-Service.
- Middleware based scheme: lo schema utilizzato per l'autenticazione tramite eIDAS-Middleware-Service.

Requisiti per gli stakeholder coinvolti:

- Relying party: richiede autenticità, integrità e riservatezza (per gli obblighi di protezione dei dati) dei dati ricevuti che identificano una persona.
- Cittadino: si aspetta riservatezza dei propri dati e privacy all'interno della rete eIDAS.
- Componente di eIDAS-Network: i requisiti sono quelli richiesti dai Relying Parties (RP) e dai cittadini.

2.6.2 eIDAS-Node

Gli eIDAS-Connector devono essere in grado di richiedere gli attributi obbligatori e facoltativi richiesti dal receiving MS per conto di terze parti, mentre gli eIDAS-Services devono essere in grado di rispondere con almeno gli attributi obbligatori previsti dal Minimum Data Set (MDS) dello schema eID del sending MS, quelli facoltativi invece, possono essere supportati o meno, in quest'ultimo caso vengono ignorati.

2.6.3 Messaggi scambiati

I nodi eIDAS utilizzano SAML per qualsiasi tipo di comunicazione, aggiungendo ai messaggi anche ulteriori informazioni necessarie come URL e certificati presenti nei metadati SAML.

I messaggi di richiesta SAML devono essere firmati e utilizzare HTTP redirect binding o HTTP POST. Ogni messaggio di richiesta deve essere verificato, in autenticità ed integrità, dall'eIDAS-Service prima di essere elaborato: il nodo di servizio ottiene il certificato del nodo connettore dall'oggetto metadati SAML del messaggio, quindi verifica la firma del messaggio stesso. Se il processo fallisce e il messaggio non può essere verificato, viene rifiutato automaticamente.

Lo stesso flusso può essere applicato anche ai messaggi di risposta, dai nodi di servizio fino a quelli di connessione. In questo caso il messaggio deve contenere uno ed un solo elemento cifrato nell'assertion contenente un AuthStatement ed un AttributeStatement. Se l'assertion è firmata, anche la sua firma deve essere verificata dal nodo connettore che riceve il messaggio di risposta.

Sempre prima di verificare una firma, il verificatore esegue una convalida dello schema XML dell'oggetto ricevuto. Nel caso di assertion crittografata, la convalida dello schema viene eseguita dopo la decrittografia. I messaggi di risposta non richiesti non vengono accettati.

2.6.4 Certificati e metadati

Tutti i MSs devono scambiarsi reciprocamente i trust anchor sotto forma di certificati per riconoscere come validi tutti i certificati utilizzati dal connettore e dai nodi di servizio. In particolare, l'oggetto di metadati SAML deve essere firmato e includere una catena di certificati a partire dal trust anchor precedentemente citato. Questi metadati SAML devono essere disponibili pubblicamente sotto URL HTTPS predefiniti. L'URL è infatti presente nel campo ¡Issuer¿ delle richieste e risposte SAML come origine da cui ottenere i metadati necessari (spesso memorizzati nella cache per evitare ritardi durante la procedura).

2.6.5 Requisiti criptografici

Per quanto riguarda i requisiti crittografici nel framework di interoperabilità, la rete eIDAS per le comunicazioni tra i nodi e i cittadini è protetta dal protocollo TLS. In particolare, l'ultima specifica tecnica adottata nel settembre 2013 riguarda l'uso di TLS 1.2 (rilasciato nel 2008) anche se una versione più recente, TLS 1.3, è stata rilasciata nel 2018. Le versioni precedenti non sono accettate.

Inoltre, i nodi eIDAS utilizzano solo suite di crittografia che forniscono perfect-forward-secrecy[2] con una chiave casuale (chiave effimera) generata da generatori di numeri crittograficamente sicuri e, ovviamente, utilizzata una sola volta per ogni trasmissione.

Infine, per quanto riguarda SAML, viene utilizzato per proteggere la riservatezza, l'autenticità e l'integrità dei dati di identificazione e l'identificazione sicura degli endpoint di comunicazione. Utilizza XML encryption e XML signatures basate sull'autenticazione tramite certificati X.509 e fornisce un livello di sicurezza di almeno 120 bit[3].

2.6.6 Flusso del processo

- All'inizio del processo, l'utente, ha la possibilità di selezionare il MS da utilizzare per la propria autenticazione, qualora non fosse già stata fornita dalla RP richiedente.
- Il nodo connettore invia una richiesta SAML al nodo di servizio corrispondente al MS selezionato. La richiesta contiene: il tipo di RP, se si tratta di un'entità non pubblica, contiene anche un RequesterID, un elemento RequestAuthContext con specificato il minimo richiesto eIDAS Level of Assurance (LoA).
- Il nodo di servizio esegue l'autenticazione della persona considerando lo schema eID selezionato e uno dei LoA richiesti.
- Infine, tornando al nodo connettore, viene effettuato il controllo finale della risposta, l'assertion viene decifrata e i dati della persona autenticata vengono inviati alla RP richiedente. Se si verifica un errore o un guasto durante uno dei passaggi del flusso del processo, la procedura viene interrotta e l'errore viene gestito seguendo le specifiche SAML.

Nel sequence diagram sopra è rappresentata parte di un vero e proprio flusso per l'autenticazione eIDAS nella sua ultima release eIDAS 2.6. È possibile notare che, oltre all'eIDAS-Connector e all'eIDAS-Service (indicato come eIDAS Proxy Service in figura), esistono altri componenti come Specific Connector, Specific Communication e Specific Proxy Service (che fa parte del sistema nel paese B e quindi non mostrato nel diagramma) che compongono quella che viene chiamata MS Specific Part del sistema nell'infrastruttura generale eIDAS.

2.6.7 MS Specific

La parte di codice specifico è responsabile della convalida delle identità digitali degli utenti a partire dalla giurisdizione locale, assicurando la conformità alle leggi nazionali e ai regolamenti in vigore in UE. Ciò consente al sistema eIDAS di fornire un approccio armonizzato e unificato alla verifica dell'identità digitale in tutta l'UE, rispettando al contempo la sovranità nazionale e i quadri giuridici di ciascuno stato membro.

Il flusso interno completo del nodo eIDAS nel sending MS, stato membro A o paese A in questo caso, viene gestito utilizzando strumenti quali Light request, Light response e Light token, utilizzati per fornire un'esperienza più efficiente e di facile utilizzo per gli sviluppatori nella versione più recente del codice eIDAS. In particolare, l'utilizzo dei token, rende disponibile un metodo più sicuro per lo scambio di informazioni tra i vari endpoint interni: i token sono direttamente correlati alle informazioni effettivamente memorizzate nella cache del nodo, condivise tra tutti i componenti del nodo. Sono necessari pochi passaggi in più per gestire questo meccanismo, ma, in questo modo, i dati non vengono effettivamente scambiati utilizzando la rete, rendendo davvero difficile per un intruso sniffarli o modificarli.

[2]Proprietà di protocolli di comunicazione sicuri che assicurano che le chiavi di crittografia, generate per una sessione, non possano essere compromesse retroattivamente anche se un utente malintenzionato ottiene l'accesso a chiavi segrete a lungo termine.

[3]Utilizza chiavi crittografiche da 120 bit.

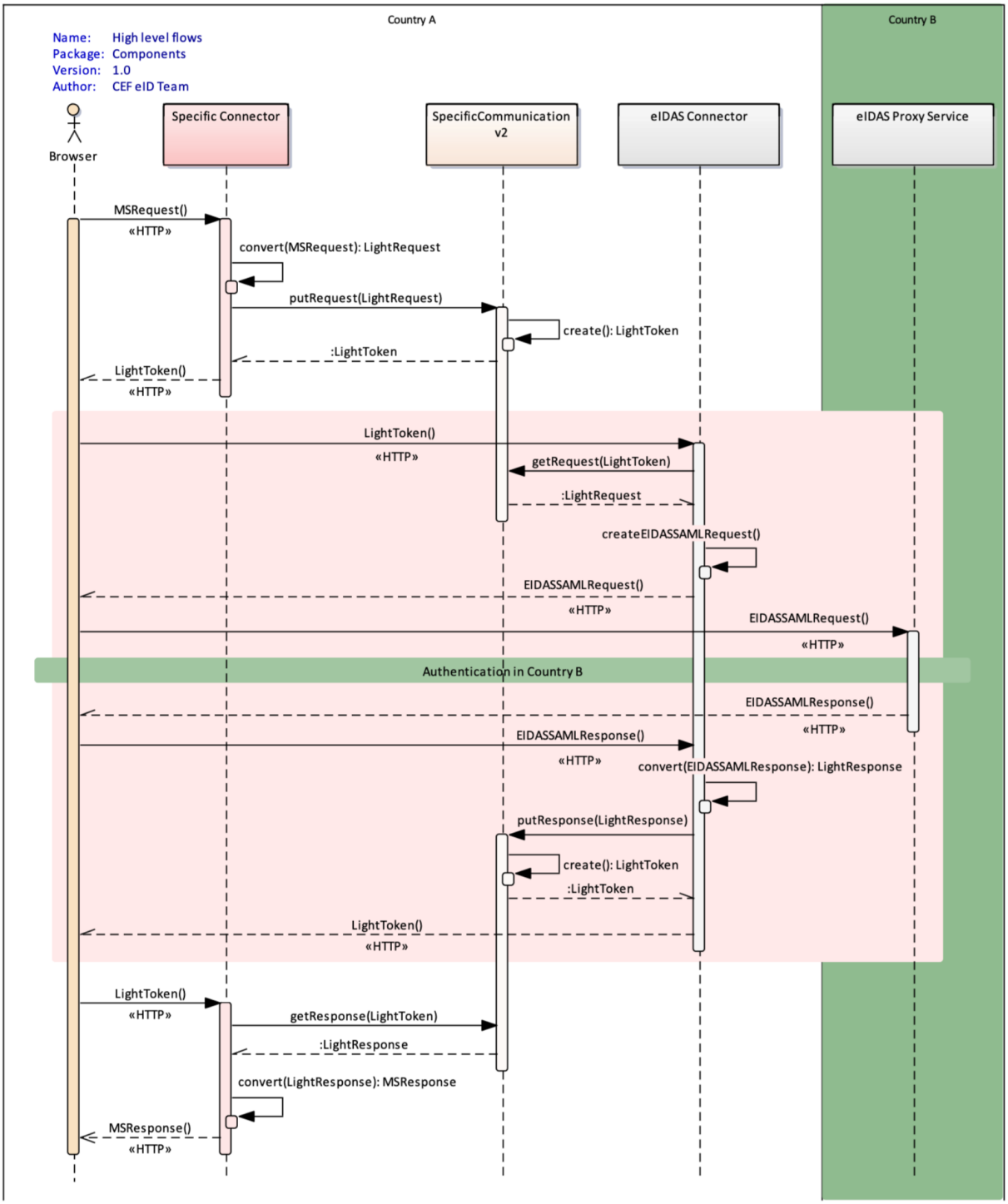

Figura 2.4. Flusso di alto livello eIDAS (source: European Commission).

Il codice eIDAS, disponibile sul sito della Commissione Europea[4], è scritto utilizzando principalmente il linguaggio JAVA e adotta il meccanismo dei servlet per gestire le varie richieste e risposte HTTP. Quando una nuova richiesta SAML di autenticazione viene emessa dall'eIDAS-Connector, viene quindi inviata all'eIDAS-Service nel paese del cittadino reindirizzando l'utente

[4] https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eIDAS-Node+version+2.6

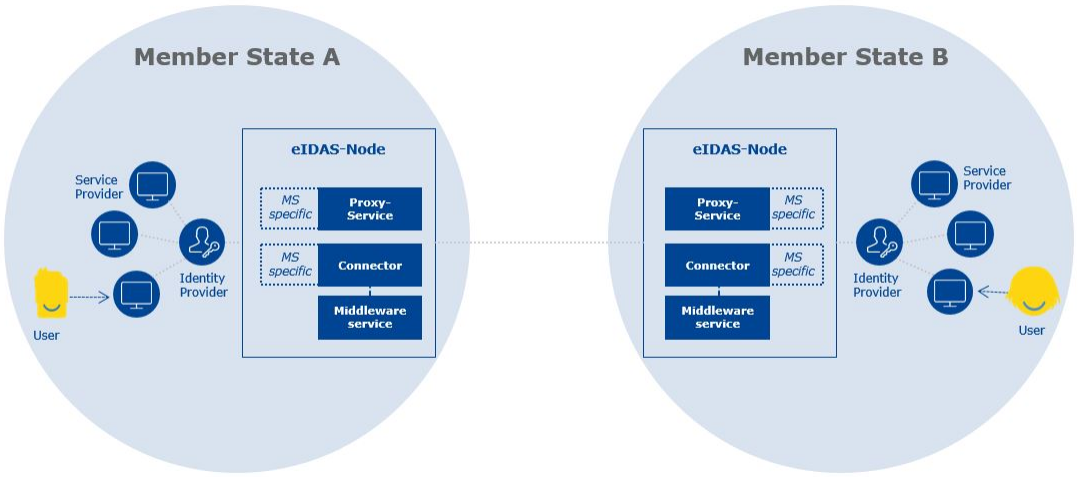

Figura 2.5. MS Specific in eIDAS (source: European Commission).

con una POST all'endpoint esterno.

2.7 Il sistema pubblico italiano per l'identità digitale (SPID)

SPID (Sistema Pubblico di Identità Digitale) è la chiave di accesso a tutti i servizi digitali della pubblica amministrazione ed a svariati del settore privato. Consiste in un singolo set di credenziali (username e password) che rappresenta l'identità digitale e personale dei cittadini italiani, consentendo loro di essere riconosciuti e di accedere in modo sicuro e personalizzato ai servizi digitali. Inoltre, SPID consente l'accesso ai servizi pubblici degli altri stati membri dell'Unione Europea e ad altre società private che lo hanno scelto come uno degli strumenti di identificazione.

Il sistema di identificazione attraverso SPID si basa su tre diversi livelli di sicurezza progressivamente più rigorosi, richiesti dai servizi durante la fase di accesso e correlati al tipo di attività che l'utente sta per svolgere. Per ottenere le credenziali SPID esistono diversi IdP, ad esempio TIM id, POSTE id, aruba.it id, e ognuno di essi può garantire al cliente un certo livello di sicurezza raggiungibile.

I tre livelli sono:

1. Accesso con semplice nome utente e password (SPIDL1).
2. Stesso del punto 1 più un codice temporaneo, un OTP (One Time Password), come ad esempio un SMS o il supporto offerto dall'applicazione mobile (per smartphone e tablet) (SPIDL2).
3. Richiede altri tipi di soluzioni di sicurezza, come un dispositivo fisico (ad esempio una smart card) rilasciato dall'identity provider (SPIDL3).

L'intero sistema SPID è oggi principalmente implementato utilizzando lo standard informatico SAML, che può essere integrato seguendo le regole per gli sviluppatori e una procedura tecnica in diverse applicazioni web, oltre a una procedura amministrativa di configurazione per l'IdP al fine di comunicare con i servizi menzionati. Un'altra possibilità offerta da SPID è di abilitare l'eIDAS login per i fornitori di servizi pubblici: in questo caso, il nodo eIDAS italiano funziona in modo simile a un IdP SPID. Ciò è obbligatorio, dal 29 settembre 2018, per le amministrazioni pubbliche che offrono servizi digitali.

2.7.1 Regole tecniche per l'integrazione di OAuth 2.0 e OIDC in SPID

Le sezioni seguenti si concentreranno sulle linee guida fornite da AGID (Agenzia per l'Italia Digitale) volte a regolamentare l'adozione di OAuth 2.0 e OIDC nel sistema SPID, al fine di intraprendere un approccio più moderno in termini di tecnologia e protocolli, in grado di garantire un processo adeguato alla vasta maggioranza dei casi d'uso in questo mondo digitale e connesso orientato alla mobilità.

La seguente tabella confronta i termini utilizzati con il protocollo SAML e i nuovi termini per OIDC, poiché è evidente una profonda correlazione delle funzionalità tra i due protocolli.

SAML	*OIDC*
Assertion	ID Token
Attribute query	UserInfo Endpoint
Authentication request	Authentication request
ForceAuthn	prompt=login
Identity Provider (IdP)	OpenID Provider (OP)
IdP metadata	OpenID Provider metadata
Issuer	Issuer
Logout	Revoke
NameID policy	Subject identifier type
Passive Authentication	prompt=none
Service Provider (SP)	Relying Party (RP)
SP metadata	Client metadata
Subject	Subject Identifier
Attributes	Claims

2.7.2 Metadati

I metadati sono strutture di dati contenenti informazioni sull'OpenID Provider (OP) e sulla Relying Party (RP), conservati e distribuiti dal Registro SPID a tutti i membri della federazione, al fine di consentire le configurazioni necessarie nei rispettivi sistemi.

Elementi chiave nei metadati OP:

- *issuer*: identificativo per l'OP (con schema HTTPS), tipicamente il bae URL. Deve corrispondere al valore di iss nel token ID emesso dall'OP. Corrisponde all'attributo entityID in SAML e rappresenta la chiave unica per identificare l'IdP.
- *authorization_endpoint*: URL per l'endpoint di autorizzazione, al quale il client verrà reindirizzato per avviare il flusso di autenticazione.
- *token_endpoint*: URL per l'endpoint del token che la RP utilizzerà per scambiare il codice ricevuto alla fine del processo di autenticazione con un access token.
- *userinfo_endpoint*: URL per lo user info endpoint che la RP può invocare per ottenere gli attributi autorizzati dall'utente.
- *introspection_endpoint*: URL per l'introspection endpoint che restituisce informazioni su un token.
- *revocation_endpoint*: URL per l'endpoint di revoca che revoca un refresh token o access token precedentemente emesso per la RP richiedente.
- *jwks_uri*: URL per il jwks che è un json contenente i seguenti parametri:
  - *kty*: famiglia dell'algoritmo crittografico adottato.
  - *alg*: algoritmo adottato.

```
{
  "issuer": "https://op.fornitore_identita.it",
  "authorization_endpoint": "https://op.fornitore_identita.it/auth",
  "token_endpoint": "https://op.fornitore_identita.it/token",
  "userinfo_endpoint": "https://op.fornitore_identita.it/userinfo",
  "introspection_endpoint": "https://op.fornitore_identita.it/introspect",
  "revocation_endpoint": "https://op.fornitore_identita.it/revoke",
  "end_session_endpoint": "https://op.fornitore_identita.it/logout",
  "jwks_uri": "https://registry.spid.gov.it/...",
  "id_token_encryption_alg_values_supported": ["..."],
  "userinfo_signing_alg_values_supported": ["..."],
  "request_object_encryption_enc_values_supported": ["..."],
  "token_endpoint_auth_methods_supported": ["private_key_jwt"],
  "userinfo_encryption_alg_values_supported": ["..."],
  "id_token_encryption_enc_values_supported": ["..."],
  "id_token_signing_alg_values_supported": ["..."],
  "request_object_encryption_alg_values_supported": ["..."],
  "token_endpoint_auth_signing_alg_values_supported": ["..."],
  "request_object_signing_alg_values_supported": ["..."],
  "userinfo_encryption_enc_values_supported": ["..."],
  "claims_supported":[
    "https://attributes.spid.gov.it/spidCode",
    "https://attributes.spid.gov.it/name",
    "https://attributes.spid.gov.it/familyName",
    "https://attributes.spid.gov.it/placeOfBirth",
    "https://attributes.spid.gov.it/countyOfBirth",
    "https://attributes.spid.gov.it/dateOfBirth",
    "https://attributes.spid.gov.it/gender",
    "https://attributes.spid.gov.it/companyName",
    "https://attributes.spid.gov.it/registeredOffice",
    "https://attributes.spid.gov.it/fiscalNumber",
    "https://attributes.spid.gov.it/ivaCode",
    "https://attributes.spid.gov.it/idCard",
    "https://attributes.spid.gov.it/mobilePhone",
    "https://attributes.spid.gov.it/email",
    "https://attributes.spid.gov.it/address",
    "https://attributes.spid.gov.it/expirationDate",
    "https://attributes.spid.gov.it/digitalAddress"
  ],
  "acr_values_supported":[
    "https://www.spid.gov.it/SpidL1",
    "https://www.spid.gov.it/SpidL2",
    "https://www.spid.gov.it/SpidL3
  ],
  "request_parameter_supported": true,
  "subject_types_supported":["public"],
  "op_name": "Agenzia per l'Italia Digitale",
  "op_name#en": "Agency for Digital Italy",
  "op_url": "https://www.agid.gov.it",
  "op_url#en": "https://www.agid.gov.it/en"
}
```

Figura 2.6. Esempio di metadati dell'OP.

```
{
  "keys": [
    {
     "kty": "EC",
     "kid": "sig-ec256-0",
     "use": "sig",
     "crv": "P-256",
     "x": "2jM2df3IjB9VYQ0yz373-6EEot_1TBuTRaRYafMi5K0",
     "y": "h6Zlz6XReK0L-iu4ZgxlozJEXgTGUFuuDl7o8b_8JnM"
    },
    {
     "kty": "EC",
     "kid": "enc-ec256-0",
     "use": "enc",
     "crv": "P-256",
     "x": "QI31cvWP4GwnWIi-Z0IYHauQ4nPCk8Vf1BHoPazGqEc",
     "y": "DBwf8t9-abpXGtTDlZ8njjxAb33kOMrOqiGsd9oRxr0"
    }
   ]
}
```


Figura 2.7. Esempio di risorsa jwks_uri.

  - *use*: uso previsto per la chiave pubblica, signature (sig) o encryption (enc).
  - *kid*: identificatore univoco della chiave.
  - *n*: modulo (pem standard).
  - *e*: esponente (pem standard).

- *provider_name*: nome del provider OpenID.
- *provider_url*: URL del provider OpenID.
- altri campi contenenti gli algoritmi supportati.
- *acr_values_supported*: array contenente i livelli SPID supportati dall'OP. Uno o più tra:
  - *https://www.spid.gov.it/SpidL1*
  - *https://www.spid.gov.it/SpidL2*
  - *https://www.spid.gov.it/SpidL3*

Elementi chiave nei metadati dell'RP:

- *client_id*: URI per identificare univocamente la RP, come da registro SPID.
- *redirect_uris*: elenco di URI di call-back utilizzati dalla RP. Il protocollo HTTPS è obbligatorio. Uno di essi deve essere presente nella richiesta di autenticazione.
- *jwks_uri*: stesso formato di quello presente nei metadati dell'OP.
- *client_name*: nome della RP da mostrare nelle pagine di autenticazione e consenso.
- *response_types*: deve contenere solo il valore code *code*
- *grant_types*: deve contenere solo i valori *authorization_code* e *refresh_token*

```
{
  "client_id": "https://rp.spid.agid.gov.it",
  "redirect_uris": [
    "https://rp.spid.agid.gov.it/callback1/",
    "https://rp.spid.agid.gov.it/callback2/"
  ],
  "jwks_uri": "https://registry.spid.gov.it/...",
  "jwks": {
    "keys": [
     {
      "kty": "RSA",
      "alg": "RS256",
      "use": "sig",
      "kid": "e27671d73a2605ccd454413c4c94e25b3f66cdea",
      "n": "vmyoDT6ND_YJa1ItdvULuTJr2pw4MvN3Z5kmSiJBm9glVoakcDEBGF4b5c7WDh2P
          ...",
      "e": "ABAB"
     }
    ]
  },
  "response_types": ["code"],
  "grant_types": ["authorization_code", "refresh_token"],
  "client_name": "Agenzia per l'Italia Digitale",
  "client_name#en": "Agency for Digital Italy"
}
```

Figura 2.8. Esempio di metadati della RP.

2.7.3 Flusso

Il flusso si basa sull'authorization code flow di OpenID Connect, l'unico richiesto da iGov.

L'authorization code flow restituisce un authorization code che può essere successivamente scambiato per un token ID e/o un access token. Questo tipo di flusso è anche ideale per sessioni a lungo termine o aggiornabili tramite refresh token. L'authorization code flow permette di ottenere l'authorization code dall'endpoint di autorizzazione dell'OP, mentre tutti i token emessi vengono restituiti dal token endpoint.

2.7.4 Authentication request

Per iniziare il flusso di autenticazione, la RP reindirizza l'utente all'authorization endpoint dell'OP selezionato, inviando con un HTTP GET o POST la richiesta nel formato JWT, firmato e crittografato.

Elementi chiave nella authentication request:

- *client_id:* URI per identificare in modo univoco l'RP, come da registro SPID.
- *code_challenge*: una challenge per il PKCE[5] da inviare anche nella successiva richiesta al token endpoint.

[5]Proof Key for Code Exchange è un'estensione per OAuth 2.0 al fine di evitare possibili attacchi messi in atto intercettando il codice di autorizzazione.

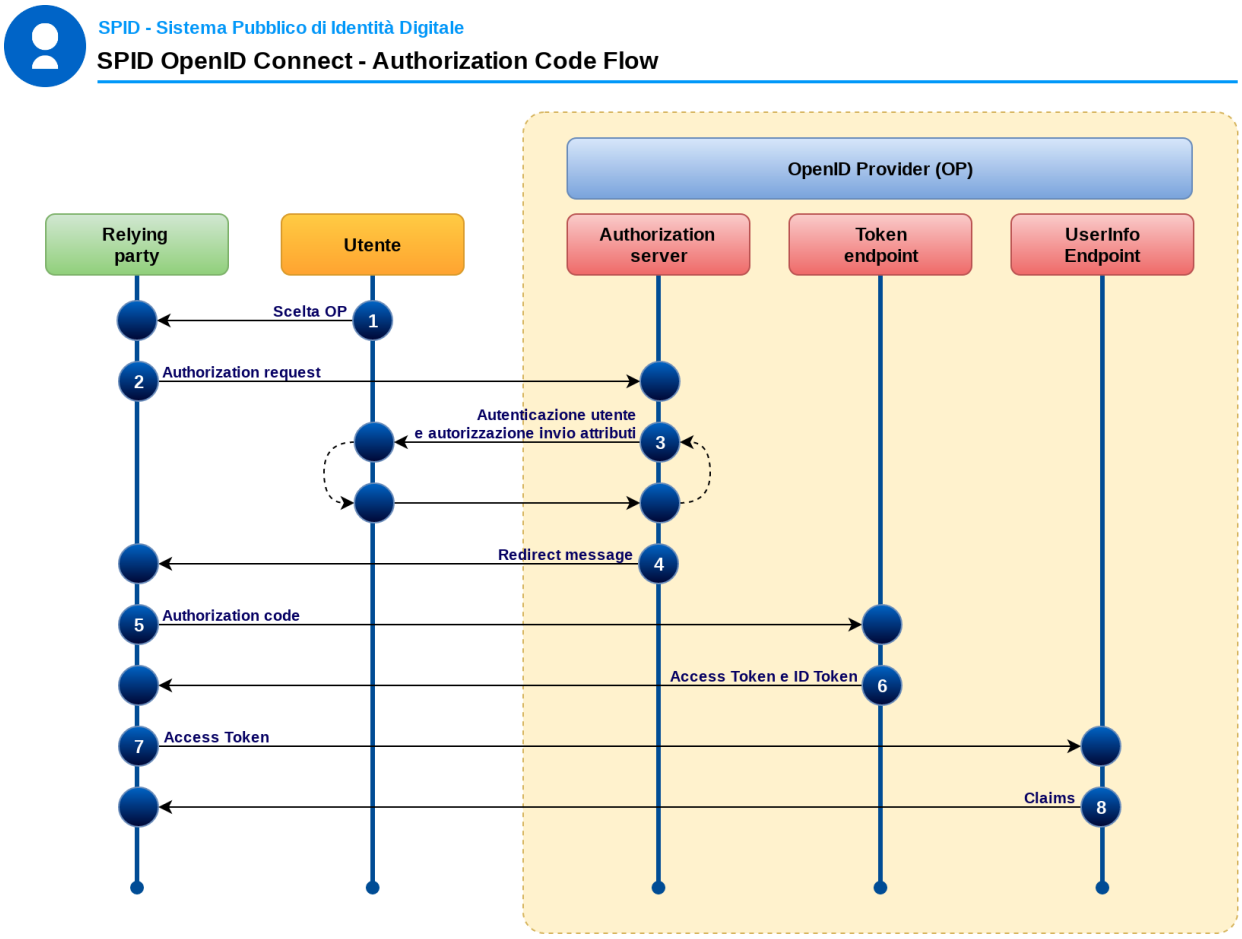

Figura 2.9. Flusso con authorization code per SPID OIDC (source: AGID).

- *code_challenge_method*: metodo per la creazione della challenge. Il valore deve essere S256 (SHA-256).
- *nonce*: valore per evitare replay attacks, generato casualmente. Questo valore verrà restituito nell'ID Token emesso dal token endpoint, per consentire al client di verificare che sia lo stesso inviato nell'authentication request.
- *prompt*: definisce se l'OP deve gestire una richiesta di autenticazione dell'utente. I possibili valori sono:
  - *consent*: l'OP chiederà le credenziali dell'utente solo se non è attiva una sessione SSO e quindi chiederà di autorizzare gli attributi richiesti.
  - *consent login*: l'OP chiederà sempre le credenziali dell'utente e poi chiederà di autorizzare gli attributi richiesti.
- *redirect_uri*: URL verso il quale l'OP reindirizzerà l'utente dopo il processo di autenticazione.
- *response_type*: tipo di credenziali che l'OP deve restituire. In questo caso, il valore deve essere *code*
- *scope*: elenco degli scope richiesti. Il valore *openid* deve essere presente, il valore *offline_access*, se presente, informerà l'OP di emettere anche un refresh token insieme all'access token. Principalmente utilizzato per i client di app mobili.
- *acr_values*: elenco di valori separati da uno spazio che specificano i valori acr richiesti come indicato nei metadati dell'OP.

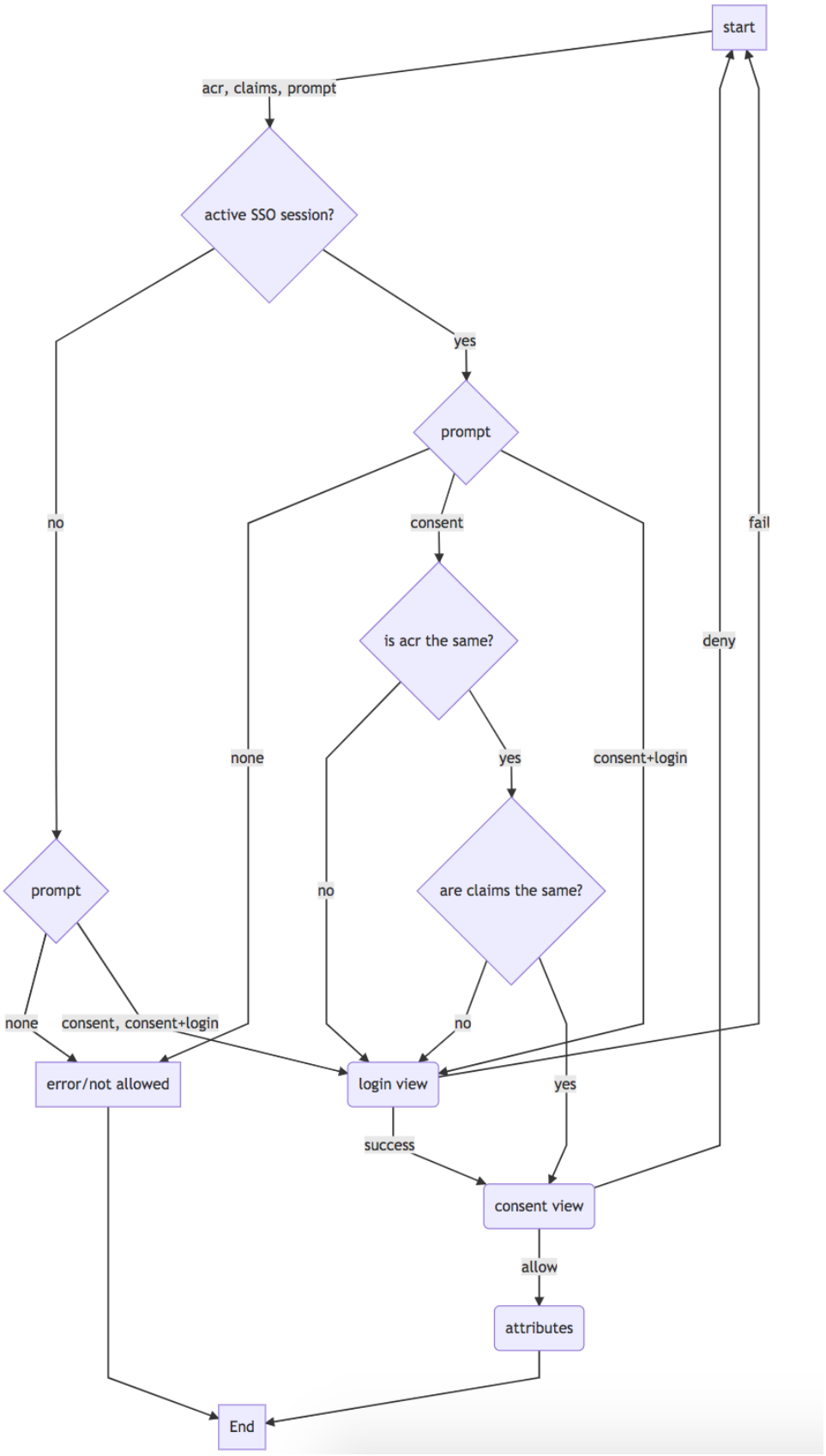

Figura 2.10. Diagramma di sessione SPID OIDC (source: AGID).

```
{
 client_id=https%3A%2F%2Frp.spid.agid.gov.it
 code_challenge=qWJlMe0xdbXrKxTm72EpH659bUxAxw80
 code_challenge_method=S256
 nonce=MBzGqyf9QytD28eupyWhSqMj78WNqpc2
 prompt=login
 redirect_uri=https%3A%2F%2Frp.spid.agid.gov.it%2Fcallback1%2F
 response_type=code
 scope=openid
 acr_values=https://www.spid.gov.it/SpidL1 https://www.spid.gov.it/SpidL2
 claims={
  "id_token":{
   "nbf": { essential: true},
   "jti": { essential: true}
  },
  "userinfo":{
   "https://attributes.spid.gov.it/name": null,
   "https://attributes.spid.gov.it/familyName": null
  },
 }
state=fyZiOL9Lf2CeKuNT2JzxiLRDink0uPcd
}
```

Figura 2.11. Esempio di authentication request, risorsa redirect_uri decodificata.

- *claims*: elenco di attributi richiesti dall'RP.
- *response_mode*: definisce il metodo per la Form response. Il valore deve essere textitform_post.

2.7.5 Authentication response

L'authentication response è un messaggio di risposta nel flusso OAuth 2.0, restituito dall'authorization endpoint dell'OP dopo il completamento del processo di autenticazione. L'OP reindirizzerà l'utente all'URI di call-back specificata nella authorization request, aggiungendo nella POST i parametri di risposta (tipicamente il code).

Gli elementi chiave nella authentication response sono:

- *code*: codice di autorizzazione unico che il client trasmetterà al token endpoint.
- *state*: il valore è lo stesso di quello contenuto nella richiesta. Il client deve verificare la corrispondenza tra i due.

In caso di errore, l'OP invia al redirect uri del client un codice di errore con una descrizione e in ogni caso il valore dello state della richiesta originale. I codici di errore sono:

- *access_denied*: l'OP ha bloccato l'accesso a causa di credenziali errate o non in linea con il livello SPID richiesto (livello acr).
- *invalid_client*: il valore client_id nella richiesta non è riconosciuto dall'OP.
- *invalid_request*: la richiesta non è valida perché uno o più parametri sono errati o mancanti.
- *server_error*: l'OP ha riscontrato un problema interno.
- *temporarily_unavailable*: l'OP ha riscontrato un problema interno temporaneo.

```
POST https://op.spid.agid.gov.it/token?
client_id=https\%3A\%2F\%2Frp.spid.agid.gov.it&
client_assertion=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWI
    iOiIxMjM0NTY3ODkwIiwibmFtZSI6IlNQSUQiLCJhZG1pbiI6dHJ1ZX0.LVyRDPVJm0S9
    q7oiXcYVIIqGWY0wWQlqxvFGYswLF88&
client_assertion_type=urn\%3Aietf\%3Aparams\%3Aoauth\%3Aclient-as sertion-type
    \%3Ajwt-bearer&
code=usDwMnEzJPpG5oaV8x3j&
code_verifier=9g8S40MozM3NSqjHnhi7OnsE38jklFv2&
grant_type=authorization_code
```

Figura 2.12. Esempio di token request utilizzando l'authorization code.

```
POST https://op.spid.agid.gov.it/token?
client_id=https\%3A\%2F\%2Frp.spid.agid.gov.it&
client_assertion=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWI
    iOiIxMjM0NTY3ODkwIiwibmFtZSI6IlNQSUQiLCJhZG1pbiI6dHJ1ZX0.LVyRDPVJm0S9
    q7oiXcYVIIqGWY0wWQlqxvFGYswLF88&
client_assertion_type=urn\%3Aietf\%3Aparams\%3Aoauth\%3Aclient-as sertion-type
    \%3Ajwt-bearer&
grant_type=refresh_token&
refresh_token=8xLOxBtZp8
```

Figura 2.13. Esempio di token request utlizzando refresh token.

2.7.6 Token endpoint

Il token endpoint emette un access token, un ID token ed un refresh token. Il cliente invia una richiesta al token endpoint includendo l'authorization code ricevuto dall'OP, per ottenere un ID token ed un access token (per richiedere attributi/claims allo userinfo endpoint) e, eventualmente, un refresh token (nel caso di una sessione a lungo termine). In quest'ultimo caso menzionato, il refresh token, viene successivamente utilizzato per ottenere un nuovo access token contattando lo stesso token endpoint e trasmettendo il refresh token invece dell'authorization code.

Elementi chiave nella richiesta del token:

- *client_id*: URI per identificare univocamente la RP, come nel registro SPID.
- *client_assertion*: JWT firmato con la chiave privata della RP, contenente i seguenti parametri:
  - *iss*: identificatore per la RP, stesso valore di cliend_id.
  - *sub*: stesso di iss.
  - *aud*: URL per il token endpoint dell'OP.
  - *iat*: data e ora in formato UTC in cui il JWT è stato creato.
  - *exp*: data e ora in formato UTC per la scadenza della richiesta.
  - *jti*: identificatore univoco per questa richiesta di autenticazione. Generato con almeno 128 bit di entropia.
- *client_assertion_type*: il valore deve essere *urn:ietf:para ms:oauth:client -assertion-type :jwt-bearer*

```
{
 "access_token": "dC34Pf6kdG...",
 "token_type": "Bearer",
 "refresh_token": "wJ848BcyLP...",
 "expires_in": 1800,
 "id_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY..."
}
```

Figura 2.14. Esempio di token response.

- *grant_type*: tipo di credenziale richiesta. Il valore deve essere uno tra *authorization_code* e *refresh_token*.
- *code*: authorization code ottenuto nella authentication response. Solo per grant_type authorization_code.
- *code_verifier*: codice per verificare il code_challenge. Codice originale di quello inviato poi con l'algoritmo SHA-256 applicato nell'authentication request come code_challenge. Solo per grant_type authorization_code.
- *refresh_token*: solo per grant_type refresh_token.

Dopo aver ricevuto e validato il token dal client, il token endpoint dell'OP, emette una risposta contenente un token ID, un access token e, eventualmente, un refresh token, in formato JWT e firmato. L'access token e l'ID token devono essere costruiti seguendo le linee guida dello standard *International Government Assurance Profile (iGov) di OAuth 2.0* relativamente ai JWT bearer token.

Elementi chiave nella token response:

- *access_token*: access token in formato JWT, firmato, consente al client l'accesso allo userinfo endpoint per ottenere gli attributi/claims.
- *token_type*: tipo di access token restituito. Il valore deve essere *Bearer*.
- *refresh_token*: refresh token in formato JWT firmato, consente al client di richiedere nuovamente un nuovo access token al token endpoint, per poi procedere con una sessione a lungo termine.
- *expires_in*: scadenza dell'access token espressa in secondi.
- *id_token*: ID token in formato JWT, firmato e cifrato.

2.7.7 ID token

Il token ID è un JWT contenente informazioni sullo user autenticato. I client devono sempre validare l'ID token.

Elementi chiave dell'ID token:

- *iss*: identifica in modo univoco l'OP, come registrato nella federazione SPID.
- *sub*: identifica il soggetto. Il valore è stabilito seguendo lo standard OpenID Connect Core, paragrafo 8.1 *Pairwise identifier Algorithm*.
- *aud*: contiene l'ID del client.

```
{
 "iss": "https://op.spid.agid.gov.it/",
 "sub": "OP-1234567890",
 "aud": "https://rp.spid.agid.gov.it/",
 "acr": "https://www.spid.gov.it/SpidL2",
 "at_hash": "qiyh4XPJGsOZ2MEAyLkfWqeQ",
 "iat": 1519032969,
 "nbf": 1519032969,
 "exp": 1519033149,
 "jti": "nw4J0zMwRk4kRbQ53G7z",
 "nonce": "MBzGqyf9QytD28eupyWhSqMj78WNqpc2"
}
```

Figura 2.15. Esempio di JWT payload di un ID token.

- *acr*: livello di autenticazione reale. Deve essere uguale o superiore a quello richiesto nella richiesta di autenticazione.
- *at_hash*: hash della prima metà dell'access token codificato in base64. Il client deve verificare che il valore sia lo stesso dell'access token restituito con il token ID.
- *iat*: data e ora di emissione del token in formato UTC.
- *nbf*: data e ora di inizio di validità del token in formato UTC. Il valore deve essere lo stesso di *iat*.
- *exp*: data e ora di scadenza del token in formato UTC.
- *jti*: identificatore univoco per il token ID che il client può verificare per prevenire le ripetizioni.
- *nonce*: stringa casuale generata dal client per ogni sessione utente e inviata originariamente nella authentication request. Anche questo serve per prevenire i reply attacks. Il client deve verificare che sia lo stesso inviato nell'authentication request.

In caso di errore, l'OP restituisce un codice di errore HTTP 401 con un JSON nel corpo con il seguente elemento:

- *error*: codice di errore. Deve essere uno dei seguenti:
  - *invalid_client*: l'ID del client nella richiesta non è riconosciuto.
  - *unsupported_grant_type*: il campo grant type contiene un valore errato.
  - *invalid_grant*: i parametri grant type, code, code verifier, access token non sono validi.
  - *invalid_request*: richiesta non valida: la richiesta non è valida perché uno o più dei parametri sono errati o mancanti.
  - *server_error*: l'OP ha riscontrato un problema interno.
  - *temporarily_unavailable*: l'OP ha riscontrato un problema interno temporaneo.
- *error_description*: descrizione più dettagliata dell'errore. Il suo scopo principale è aiutare lo sviluppatore durante la fase di debug. Questo messaggio dovrebbe essere nascosto all'utente finale.

```
GET https://op.spid.agid.gov.it/userinfo
Authorization: Bearer dC34Pf6kdG
```

Figura 2.16. Esempio di request allo userinfo endpoint.

```
{
  "iss": "https://op.fornitore_identita.it",
  "aud": "https://rp.fornitore_servizio.it",
  "iat": 1519032969,
  "nbf": 1519032969,
  "exp": 1519033149,
  "sub": "OP-1234567890",
  "https://attributes.spid.gov.it/name": "Mario",
  "https://attributes.spid.gov.it/familyName": "Rossi",
  "https://attributes.spid.gov.it/fiscalNumber": "MROXXXXXXXXXXXXX"
}
```

Figura 2.17. Esempio di risposta dallo userinfo endpoint.

2.7.8 Userinfo endpoint (attributes/claims)

L'endpoint userinfo è una risorsa protetta da OAuth 2.0 che restituisce gli attributi dell'utente autenticato. Per ottenere gli attributi richiesti dalla RP, il client invia una richiesta all'endpoint userinfo inserendo l'access token. Il risultato viene presentato in un JSON contenente un insieme di chiavi e valori.

L'endpoint userinfo deve supportare l'uso dei metodi HTTP GET e POST definiti in RFP 2616 e accettare l'access token sotto forma di bearer token OAuth 2.0 come espresso in RFC 6750, infine deve supportare CORS e/o altri metodi per consentire ai client Java Script di accedere all'endpoint.

La risposta dell'endpoint userinfo deve essere firmata e cifrata e contenere le richieste autorizzate nella richiesta di autenticazione.

Elementi chiave nella risposta dell'endpoint userinfo:

- *sub*: identificatore del soggetto. Il valore è lo stesso già rilasciato nell'ID token. La RP deve verificare che il valore sia lo stesso.
- *aud*: identificatore del destinatario della risposta, in questo caso la RP.
- *iss*: URI per identificare in modo univoco l'OP come registrato nel registro SPID.
- *attribute or claim*: tutti i claim richiesti durante l'autenticazione.

In caso di errore di autenticazione, l'endpoint userinfo restituisce un errore con il codice di stato HTTP 401.

2.7.9 Revocation endpoint (logout)

Il revocation endpoint permette alla RP di richiedere la revoca di un access token o di un refresh token già ottenuto. Lo stato del token può essere verificato inviandolo al introspection endpoint. Quando l'utente esegue il logout o, nel caso in cui la sua sessione sull'RP termini, la RP deve

invocare il revocation endpoint per revocare l'access token e eventualmente il refresh token di cui dispone.

L'OP deve revocare il token specificato nella richiesta e porre fine alla sessione SSO se ancora attiva. Altri token per gli utenti, relativi ad altre sessioni, devono rimanere validi.

La richiesta al revocation endpoint deve contenere almeno il token che deve essere revocato e una client assertion, simile a quella nella richiesta al token endpoint. La risposta invece contiene sempre un codice di stato HTTP 200, anche se il token non esiste o è già stato revocato, per non fornire ulteriori informazioni al riguardo.

2.7.10 Sessioni revocabili a lungo termine

Nelle applicazioni mobili in cui la RP intende garantire un'esperienza utente in cui non è necessario inserire le credenziali SPID ad ogni accesso, è possibile utilizzare sessioni a lungo termine revocabili. Per adottare questo tipo di sessione, la RP deve impostare nella authentication request lo scope *offline_access* per ottenere successivamente anche un refresh token dopo il consenso esplicito dell'utente.

1. Quando l'utente effettua l'accesso per la prima volta, deve essere avvertito della possibilità di utilizzare sessioni a lungo termine revocabili, per mantenere attiva l'autenticazione SPID di livello SPIDL1. In questo modo l'app può ricevere notifiche o effettuare azioni richieste dalla RP, anche se l'utente non è presente.
2. Le applicazioni mobili che adottano il meccanismo long-term revocable session devono richiedere all'utente, ad ogni avvio o attivazione dell'app, un PIN locale o un fattore biometrico.
3. Durante l'installazione e la prima configurazione, l'app richiede all'utente la registrazione di un fattore di autenticazione da utilizzare ogni volta.
4. Quando l'utente avvia nuovamente l'app, questa deve richiedere il fattore di autenticazione precedentemente scelto e quindi acconsentire all'accesso alle funzionalità della RP disponibili con un livello SPID 1.
5. Nel caso in cui sia necessario accedere ad alcune funzionalità che richiedono un livello SPID superiore a 1, l'utente deve effettuare un'altra autenticazione in base al livello richiesto.

Infine, l'OP deve includere una pagina raggiungibile dall'utente che mostri loro le attuali long-term session attive con la possibilità di revocarle. In caso di modifica della password, l'OP deve inoltre fornire la possibilità di revocare tutte le attuali long-term session attive dell'utente.

Capitolo 3

Metodologia

3.1 Motivazioni

Negli ultimi anni, l'interesse per le comunicazioni digitali nella rete globale internet, si è focalizzato, per quanto concerne gli standard di formato e di scambio delle informazioni tra client e server, su tecnologie come il JSON e le RESTful API. Il motore di spinta in questa direzione è stato fornito dalla necessità di gestire autorizzazione ed autenticazione, fornitura di servizi e condivisione di risorse seguendo delle linea guida comuni di facile adozione per ogni tipologia di piattaforma web, ma anche e soprattutto mobile ed IoT.

L'obiettivo è quello di predisporre un sistema efficiente, sicuro, interoperabile e scalabile da impiegare per una moltitudine di intergrazioni tra sistemi IT di natura simile o anche completamente diversa. In questo'ottica, gioca un ruolo chiave anche l'avanzamento tecnologico delle modalità per la gestione del business online, con la diffusione sempre maggiore di pagamenti in forma digitale, portafogli digitali e criptovalute per una quantità crescente di servizi offerti digitalmente da enti pubblici e privati, con target appartenenti ad ogni range di classificazione e persino target virtuali.

La filosofia con la quale lo standard OpenID Connect è stato sviluppato può essere racchiusa all'interno della frase "Rendi semplici le cose semplici, rendi possibili le cose difficili", in cui è evidente la necessità di creare uno strumento di facile implementazione ed utilizzo che allo stesso tempo fosse in grado di garantire dei livelli di avanzamento tecnologico adeguati. L'AGID (Agenzia per l'Italia Digitale) sta infatti attualmente investendo risorse per la diffusione di linee guida e applicazioni di demo per gli IdP di tipo SPID in Italia, in quanto il numero di paesi europei che adottano OIDC come standard per i relativi sistemi di identità digitale sta seguendo un trend in continua crescita.

3.2 Punti chiave dell'integrazione

3.2.1 Approccio tecnologico

L'adozione di protocolli come SAML o OAuth 2.0 con OIDC, si pone come obiettivo finale unico il raggiungimento di un medesimo risultato che garantisca un sistema sicuro permettendo autorizzazione, autenticazione e successiva fornitura di servizi o condivizione di risorse. Tuttavia, il processo con il quale l'obiettivo sopracitato venga raggiunto è caratterizzato da elementi ben differenti tra i due approcci.

Un esempio eclatante per dimostrare questa differenza, oltre l'utilizzo di JSON al posto di XML per l'adozione di OIDC in confronto a SAML, è la necessità di esporre diversi endpoint per garantire un flusso di tipo OAuth. Questo comporta ovviamente la necessità di rivedere l'architettura di un sistema che precedentemente era basato su SAML, per ognuna delle entità coinvolte, sia SP che IdP, anche se lo standard alla base per lo scambio dei messaggi sarà comunque HTTP/HTTPS.

3.2.2 User experience

Dal punto di vista dell'esperienza utente, un passaggio tecnologico come quello da SAML a OAuth 2.0 con OIDC, dovrebbe risolversi in modo quasi del tutto trasparente. I sistemi alla base che gestiscono l'autorizzazione, autenticazione, fornitura di servizi e condivisione di risorse, dovrebbe infatti garantire che la transizione non comporti nuove modalità di funzionamento o anche solo di approccio per l'utente utilizzatore.

In quest'ottica, la nuova tecnologia potrebbe addirittura apportare vantaggi riducendo la latenza per la fruizione dei sistemi e procurando nuove semplificazioni come ad esempio la possibilità di utilizzare sessioni a lungo termine con i refresh token propri di OAuth 2.0 e fornire un maggior controllo sugli accessi e sulla condivisione delle risore OIDC da parte dello user con il meccanismo di revoca dei token.

3.2.3 Sicurezza e regolazioni internazionali

Ogni nuova implementazione adottata deve rispettare le direttive e linee guida stabilite dai governi locali e dalla Commissione Europea, in termini di sicurezza e rispetto delle leggi, per essere approvata e ritenuta affidabile e applicabile. Le linee guida fornite da AGID per l'implementazione di SPID basata su OIDC, seguono infatti, a loro volta, le specifiche del profilo iGov (International Government Assurance Profile) per OIDC, che garantisce i massimi standard di sicurezza ottenibili dal protocollo al fine di permettere ai governi e agli enti privati una modalità ceritificata per l'accesso federato ai servizi pubblici online.

3.3 Set-up ambiente virtuale

La configurazione dell'ambiente virtuale simula la divisione dei ruoli tra i vari sistemi coinvolti. eIDAS e SPID, infatti, condividono e comunicano sulla stessa rete locale che rappresenta il network di internet in Europa e utilizzano, per conoscere i relativi indirizzi, la configurazione locale del file hosts, che così agisce come una sorta di DNS pubblico.

I servizi vengono eseguiti per scopi didattici su di una macchina virtuale VirtualBox, dotata del sistema operativo Ubuntu in versione 22.04.

3.3.1 eIDAS 2.6

Il sistema eIDAS è disponibile al download dal sito della Commissione Europea[1], nella sua ultima attuale versione 2.6 rilasciata il 15 Aprile 2022, insieme alla documentazione utile per l'installazione e configurazione, la migrazione da versioni più vecchie e la gestione dei codici di errore e dei log.

Il codice è scritto utilizzando principalmente il linguaggio JAVA, con file di configurazione in XML, e adotta il meccanismo dei servlet per gestire le varie richieste e risposte HTTP. Quando una nuova richiesta SAML di autenticazione viene emessa dall'eIDAS-Connector, viene quindi inviata all'eIDAS-Service nel paese del cittadino reindirizzando l'utente con una POST all'endpoint esterno (nella versione di demo fornita al download si tratta di un endpoint e un IdP interno).

Il pacchetto di eIDAS scaricato contiene al suo interno, oltre al codice JAVA, le configurazioni necessarie per l'implementazione su diversi web server. Nel caso specifico è stato adottato il web server Tomcat, versione 9, con i componenti di eIDAS corrispondenti a SP, IdP di demo, Specific Connector, Specific Proxy Service e eIDAS-Node, deploiati come file WAR (Web Application ARchive).

[1] https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eIDAS-Node+version+2.6

La versione di JAVA utilizzata, per massima compatibilità, è la 11-JDK, a cui è necessario aggiungere anche la configurazione per la librearia Bouncy Castle Provider che permette le operazioni di firma, verifica, cifratura e decifratura dei dati.

L'intero progetto JAVA è gestito mediante Maven project che è necessario installare per ricompilare e produrre i nuovi file WAR, corrispondenti ai servlet sopracitati, in caso di modifiche al codice. In particolare, è possibile eseguire il seguente comando all'interno della cartella root *EIDAS-Sources-2.6.0*

```
mvn clean install -f EIDAS-Parent/pom.xml -P {NodeOnly, DemoToolsOnly}
```

in cui è possibile scegliere tra *NodeOnly* per ottenere nella cartella *EIDAS-Node/target* il file *EidasNode.war*; *DemoToolsOnly* per ottenere in in *EIDAS-IdP-1.0/target*, *EIDAS-SP/target*, *EIDAS-SpecificConnector/target* e *EIDAS-SpecificProxyService/target* rispettivamente i file *IdP.war*, *SP.war*, *SpecificConnector.war* e *SpecificProxyService.war*.

Ottenuti quindi i file .war è possibile inserirli all'interno della cartella *webapps* di Tomcat per deploiare il relativo componenete modificato.

Un passaggio fondamentale è infine quello di aggiungere al file che gestisce il servizio Tomcat le variabili d'ambiente riferite alla versione di JAVA, alla libreria Bouncy Castle Provider ed infine ai vari file di configurazione di eIDAS, perlopiù scritti in XML e conteneti per l'appunto i riferimenti per parametri che i servlet JAVA e le relative pagine JSP popolano a RUNTIME durante l'esecuzione.

Un'esempio di file di configurazione sul quale è stato necessario apportare modofiche è quello che contiene l'URL dell'IdP verso il quale, il componente SpecificProxyService, deve rivolgersi per avviare il processo di autenticazione e autorizzazione degli attributi dell'utente. In particolare, il file *eidas-attributes.xml* presente nella cartella *EIDAS-Sources-2.6.0/EIDAS-Config/tomcat/specificProxyService* è configurato come segue al momento del download

```
<entry key="idp.url">http://localhost:8080/IdP/AuthenticateCitizen</entry>
```

indicando l'utilizzo dell'IdP di demo, corrisponente al servlet IdP, fornito da eIDAS. Il valore è stato cambiato in

```
<entry key="idp.url">http://trust-anchor.org:8000/oidc/op/authorization/</
    entry>
```

affinchè lo SpecificProxyService potesse indirizzare la richiesta di autenticazione e autorizzazione degli attributi al sistema SPID con OIDC, ed in particolare all'OP (OpenID Provider) all'URI che espone l'authorization server.

Infine, è stato inserito un identificatore per localhost, *eidas.org* all'interno del file */etc/hosts* della macchina virtuale linux, per raggiungere i servlet di Tomcat relativi al sistema SPID. Il servizio Tomcat è stato esposto quindi sulla porta *8080*.

3.3.2 SPID with OIDC (AGID)

La community di sviluppatori del governo italiano, Developers Italia, per conto di AGID (Agenzia per l'Italia Digitale) rende disponibile su GitHub una repository [2] per il download di una OpenID Connect Federation nella forma di una suite di applicazioni Django (che utilizza codice in linguaggio Python) designate per costruire una federazione OIDC di demo al fine di permettere di testare ed integrare i nuovi protocolli basati su OAuth 2.0 e OIDC per gli IdP e gli SP italiani decisi a procedere seguendo questa linea di ammodernamento tecnologico.

All'interno del pacchetto scaricato sono presenti tre progetti indipendenti, che racchiudono un determinato set di applicazioni per garantire il corretto funzionamento in funzione delle necessità. I tre progetti sono:

[2] https://github.com/italia/spid-cie-oidc-django

- federation_authority;
- relying_party;
- provider;

In particolare, il progetto di federation_authority, si basa su tutte le applicazioni necessarie per fini di sviluppo, agendo come un'entità per gestire l'intera federazione, una Relying Party SPID e un OpenID Provider SPID. Di conseguenza, il focus è stato rivolto principalmente alla configurazione e all'utilizzo di questo progetto, al fine di produrre un sistema completo e controllabile sotto ogni punto di vista per garantire una corretta integrazione con il sistema eIDAS.

A seguito della configurazione iniziale di base, effettuabile tramite il comando

```
bash docker-prepare.sh
```

che predispone i file YAML e JSON, contenuti nella paccheto di demo scaricato, per un corretto avvio delle applicazioni sotto forma di container, network e volumi basati su micorservizi Docker, è possibile, in base alle necessità personalizzare i dati di esempio, per user, rp e op, al path *spid-cie-oidc-django/examples-docker/federation_authority/dumps/example.json*, quindi sfruttando il framework di Docker-Compose, avviare il progetto con il seguente comando

```
sudo docker-compose up
```

Gli unici step preliminari per completare questa configurazione, dato che l'architettura a microservizi containerizzata offre già tutte le funzionalità di cui richiedono i progetti, riguardano quindi l'installazione sulla macchina virtuale linux di Docker (versione 4.15) e Docker Compose, in versione v1 per garantire migliore compatibilità con il sistema scaricato.

Infine, il file hosts, alla riga che definisce gli alias per il localhost, è stato arricchito con i seguenti domini

```
trust-anchor.org relying-party.org cie-provider.org
```

per permettere la raggiungibilità tra le varie applicazioni del progetto federation_authority. Il servizio OIDC di SPID è quindi raggiungibile alla porta 8000.

Il progetto di federation_authority offre, inoltre, altri strumenti per la gestione della federazione tra cui:

- Un tool per la decodifica dei JWT raggiungibile al path `trust-anchor.org:8000/onboarding/tools/decode-jwt`.
- Gli schema per i vari endpoint raggiungibili al path `trust-anchor.org:8000/onboarding/schemas/*`.
- Una dashboard di gestione dell'intera federation authority (gestione di RP, OP e relativi metadata/chiavi private, pubbliche, sessioni OIDC, token emessi, users e relativi attributi) raggiungibile al path `trust-anchor.org:8000/admin`.

3.4 Design della soluzione di integrazione

3.4.1 MS Specific

La soluzione proposta si concentra principalmente sulla modifica della parte di codice eIDAS denominata MS Specific (Member State Specific) che risiede all'interno della componenete Proxy Service di ogni nodo eIDAS nazionale. La sua funzione è quella di fornire un meccanismo di traduzione per trasformare le informazioni provenienti dal sistema eIDAS, in un formato adeguato e risolvibile dai fornitori d'intentià locali.

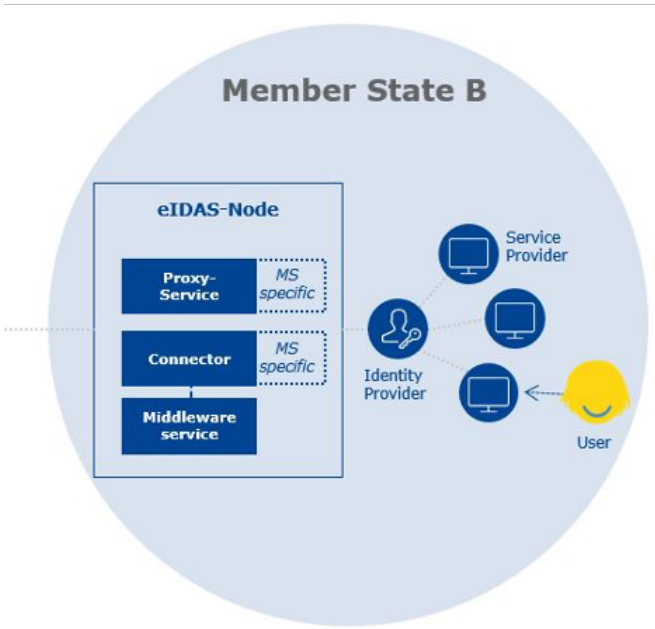

Figura 3.1. In evidenza la MS Specific part per il Proxy Service del nodo eIDAS.

In particolare, la richiesta effettuata dal SP, viene indirizzata al sistema eIDAS, sotto forma di assertion SAML. Il compito quindi del MS Specific, Specific Proxy Service, è quello di tradurre in compatibilità con il formato JSON con il fine di produrre una authentication request propria del flusso OAuth 2.0 e OIDC, in modo che l'IdP ricevente, in questo caso il sistema SPID con OIDC, possa riceverla ed interpretarla correttamente.

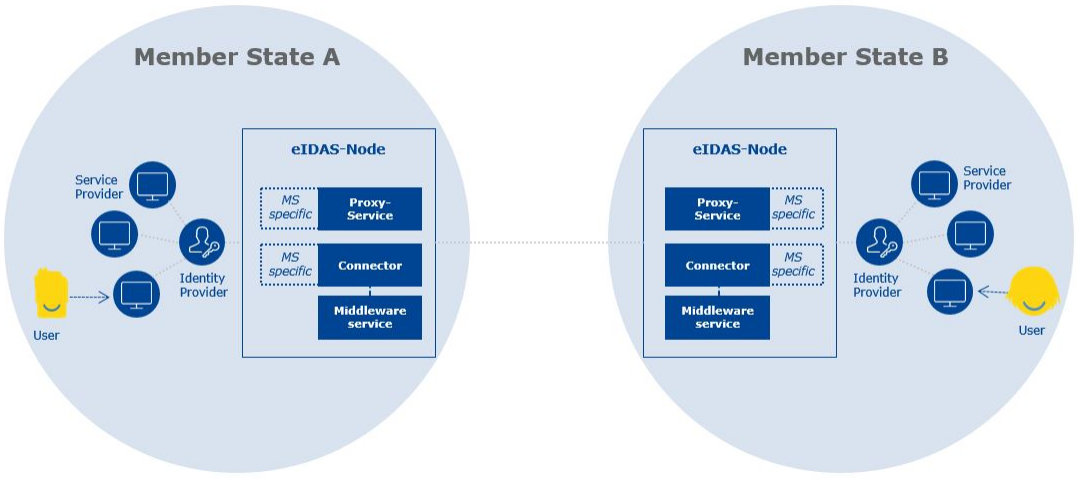

Figura 3.2. In evidenza la MS Specific part per il Connector del MS A e il Proxy Service del MS B.

Il sistema SPID con OIDC funge quindi da fornitore d'identità per il paese d'origine (B) dello user che richiede il servizio ad un SP stabilito in un paese europeo A. La vera modifica deve quindi essere applicata alla parte di codice specifico del Proxy Service del nodo eIDAS nello stato d'origine B, che in questo caso rappresenta ciò che viene denominato **sending MS**. Il sending MS riceve quindi l'autenthication request dalla componente Connector del nodo eIDAS del paese A, definito **receiving MS**.

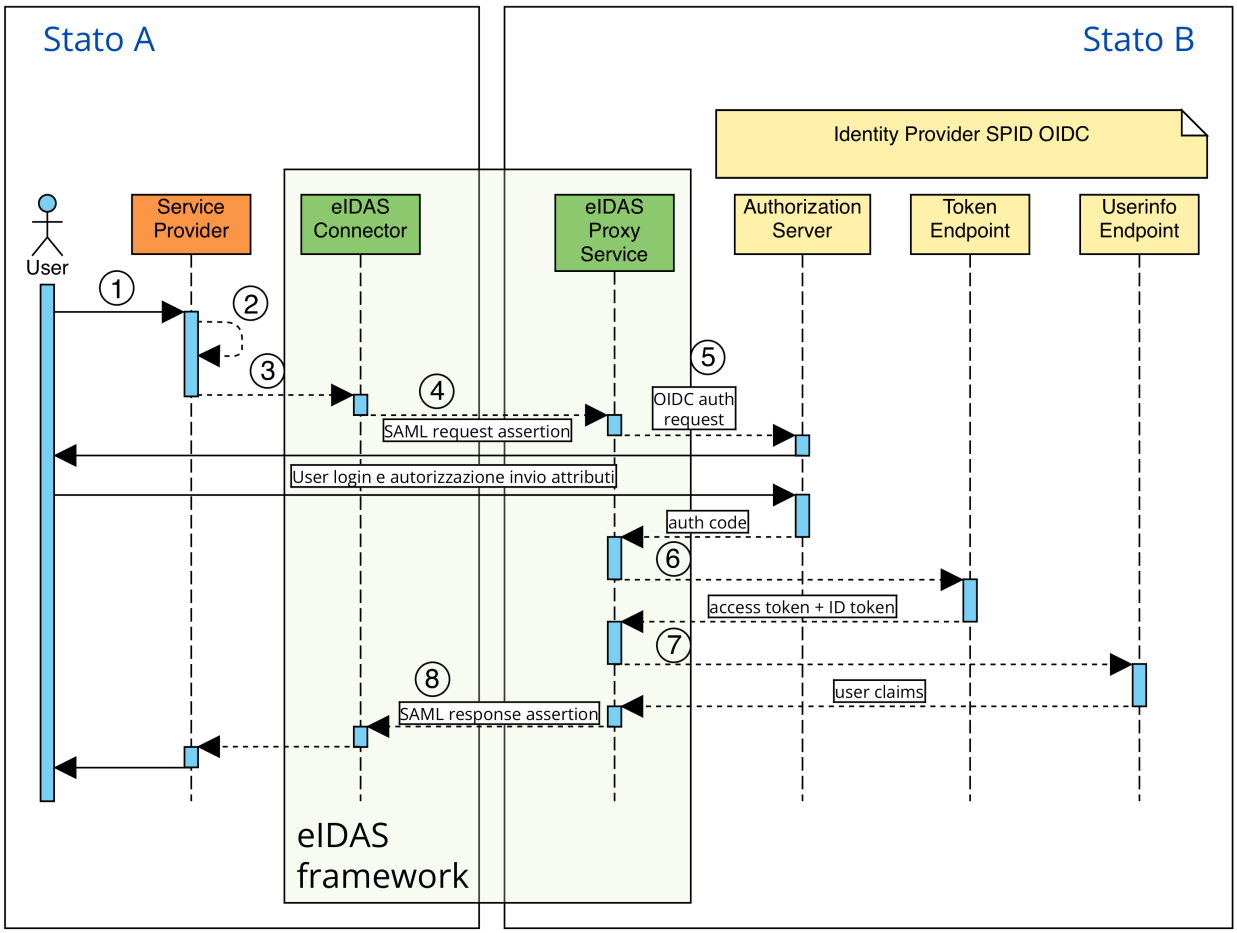

Figura 3.3. Flusso integrato tra eIDAS con Specific Proxy Service come RP e l'OP (SPID OIDC).

3.4.2 Flusso

1. Il processo viene avviato dallo user, con paese di origine B, che vuole accedere ai servizi offerti da un SP localizzato in un altro paese europeo A.

2. Il **SP** sarà quindi il service provider di demo offerto dal sistema eIDAS, localizzato in un paese A. Tale SP, sotto richiesta dell'utente, ha la necessità di autenticare lo stesso e per tale motivo produce una richiesta di autenticazione originale, configurata con gli attributi di cui necessità relativi all'utente richiedente.

3. La richiesta viene quindi inoltrata alla componente Connector del nodo eIDAS nel paese A. In particolare, la parte specifica per stato membro del Connector, trasforma e traduce la richiesta ricevuta dal SP, in un formato conforme all'utilizzo nel framework internazionale fornito da eIDAS, producendo una assertion request SAML.

4. L'assertion prodotta transita attravero il sistema di nodi eIDAS, fino a giungere alla componente Proxy Service del nodo eIDAS nel paese B, paese d'origine dell'utente richiedente. La parte specifica, Specific Proxy Service, trasforma quindi l'iniziale richiesta SAML, in formato adatto all'invio al fornitore d'identità nazionale locale (paese B). Lo Specific Proxy Service, agirà quindi, come la **RP** per il sotto flusso OAuth 2.0 con OIDC.

5. Lo Specific Proxy Service produrrà una authentication request in formato OIDC che verrà trasmessa al sistema SPID OIDC. L'IdP SPID fungerà quindi da **OP**, interpretando la richiesta e permettendo allo user richiedente di effettuare il login di autenticazione. Fornirà poi in risposta per la RP, lo Specific Proxy Service di eIDAS, un authorization code proprio del meccanismo OAuth 2.0.

6. Nuovamente, lo Specific Proxy Service, utilizzerà il codice ricevuto per ottenere, dal token endpoint dell'OP, un access token.

7. Con l'access token indirizzerà quindi una richiesta allo userinfo endpoint dell'OP per ottenere infine gli attributi richiesti inizialmente dal SP.

8. La risposta procederà quindi, dopo essere stata nuovamente convertita in assertion response SAML, verso il framework di eIDAS, giungendo al Connector del paese A e quindi infine al SP dello stesso paese che potrà utilizzare gli attributi ricevuti per permettere allo user richiedente di accedere alle relative risorse/servizi.

3.5 Dettagli implementativi

3.5.1 Passaggi preliminari

Prima di avviare il processo, risulta necessario per la RP (eIDAS Specific Proxy Service) e l'OP (SPID OIDC), iscriversi e registrarsi reciprocamente ad una federazione. Questa federazione, indicata nel caso specifico come *Registro SPID*, permette di ottenere tutti i **metadata** necessari alle due entità per lo scambio seguente di informazioni, fornendo certificati, endpoint ed altre informazioni relative. La RP viene quindi così registrata come client per l'IdP.

Per il caso specifico, sono stati utilizzati i metadata del RP di demo fornito dal sistema SPID, attribuendoli allo Specifc Proxy Service di eIDAS. Solo il parametro della *redirect_uri* è stato modificato per permettere al servlet di eIDAS di ricevere l'authorization code rilasciato dall'OP.

Tuttavia, è possibile configurare a in base alle necessità ogni parametro del RP e dell'OP, registrando anche delle nuove identità e relativa interconnessione alla federazione, grazie al progetto di federation_authority scelto per implementare il sistema SPID OIDC.

Metadati della RP:


```
{
   "openid_relying_party":{
      "organization_name":"that fancy RP",
      "application_type":"web",
      "client_id":"http://trust-anchor.org:8000/oidc/rp/",
      "client_registration_types":[
         "automatic"
      ],
      "jwks":{
         "keys":[{
               "kty":"RSA",
               "use":"sig",
               "n":"5s4qi1Ta-sEuKb5rJ8TzHmyGKaSu89pIXIi6w4Ekx6GL56mJDNE[...]
                   hDF_0kCp44UMmS3Q",
               "e":"AQAB",
               "kid":"2HnoFS3YnC9tjiCaivhWLVUJ3AxwGGz_98uRFaqMEEs"}]},
      "client_name":"Name of an example organization",
      "contacts":[
         "ops@rp.example.it"
      ],
      "grant_types":[
         "refresh_token",
         "authorization_code"
      ],
      "redirect_uris":[
         "http://localhost:8080/SpecificProxyService/oidc/callback"
      ],
      "response_types":[
```

```
            "code"
         ],
         "subject_type":"pairwise"
      }
   }
```

3.5.2 Authentication request da eIDAS a SPID

Lo Specific Proxy Service, dopo aver ricevuto la richiesta in formato SAML, la trasforma come indicato nell'esempio sotto. Tale richiesta veniva poi codificata in Base64 e inviata come parametro *SMSSPRequest* all'IdP. In particolare, nell'attribute list, sono presenti tutti gli attributi richiesti dal SP, *LegalName* e *Familyname*, mostrati nell'esempio, e altre informazioni utili di dettaglio.

```
{
   "authentication_request" : {
      "attribute_list" : [ {
         "type" : "requested_attribute",
         "name" : "LegalName",
         "required" : true
      }, [...] {
         "type" : "requested_attribute",
         "name" : "FamilyName",
         "required" : true
      } [...] ],
      "requested_authentication_context" : {
         "comparison" : "minimum",
         "context_class" : [ "B" ],
         "non_notified_context_class" : [ ]
      },
      "citizen_country" : "CA",
      "created_on" : "2023-02-12T19:34:07.854+01:00",
      "force_authentication" : true,
      "id" : "b14ffd42-6e34-4b46-8caf-268bc0d8fcd7",
      "provider_name" : "DEMO-SP-CA",
      "serviceUrl" : "http://localhost:8080/SpecificProxyService/IdpResponse\n
          ",
      "sp_type" : "public",
      "version" : "1"
   }
}
```

Con le modifiche apportate allo Specific Proxy Service, la nuova authentication request assume una forma simile a quella proposta in basso al fine di essere conforme alle richieste dell'OP SPID. Viene messo in atto un mapping dei valori per gli attributi richiesti che vengoni inseriti quindi nella sezione userinfo, producendo infine un JSON che ha come payload quello dell'esempio.

```
{
   "iss":"http://trust-anchor.org:8000/oidc/rp/",
   "scope":"openid",
   "redirect_uri":"http://localhost:8080/SpecificProxyService/oidc/callback",
   "response_type":"code",
   "nonce":"DRUILeKnIXZQ36DnfYxNDCIIYvNs5bET",
   "state":"4EfbnIVptSgz9i3m1fcO5nrr0MWd5R77",
   "client_id":"http://trust-anchor.org:8000/oidc/rp/",
   "endpoint":"http://trust-anchor.org:8000/oidc/op/authorization",
   "acr_values":"https://www.spid.gov.it/SpidL1",
   "iat":1676234010,
```

```
  "exp":1676234070,
  "jti":"d967ec6f-932c-411d-963b-7336ef08b7a0",
  "aud":[
     "http://trust-anchor.org:8000/oidc/op/",
     "http://trust-anchor.org:8000/oidc/op/authorization"],
  "claims":{
     "id_token":{
        "https://attributes.spid.gov.it/familyName":{
           "essential":true},
        "https://attributes.spid.gov.it/email":{
           "essential":true}},
     "userinfo":{
        "https://attributes.spid.gov.it/name":null,
        "https://attributes.spid.gov.it/familyName":null,
        "https://attributes.spid.gov.it/email":null,
        "https://attributes.spid.gov.it/fiscalNumber":null}},
  "prompt":"consent login",
  "code_challenge":"dl8b8CXRp6CxGviKzDvYW3ObwVm5ffhNUPXuMbGBRGU",
  "code_challenge_method":"S256"
}
```

Header e payload vengono quindi codificati in Base64 e separati da un ".", successivamente, dopo un secondo ".", viene inserita la firma digitale realizzata su header.payload con la chiave privata RSA del RP (lo Specific Proxy Service). Infine, il JWT così ottenuto, in forma di JWS, viene inviato all'authorization endpoint dell'OP come parametro titolato *request*.

Metodi JAVA per la creazione authentication request per il provider OIDC come JWS firmato con la chiave privata del RP:

```
1  httpServletRequest.setAttribute("client_id", "http%3A%2F%2Ftrust-anchor.org%3
       A8000%2Foidc%2Frp%2F");
2  httpServletRequest.setAttribute("scope", "openid");
3  httpServletRequest.setAttribute("response_type", "code");
4  String code_verifier = generateRandomString(43);
5
6  httpServletRequest.setAttribute("code_challenge", generateCodeChallenge(
       code_verifier));
7  httpServletRequest.setAttribute("code_challenge_method", "S256");
8  String nonce = generateRandomString(32);
9  String state = generateRandomString(32);
10
11 String payload = generatePayload(nonce, state, generateCodeChallenge(
       code_verifier), jsonObjectRequest);
12 String header = "{\"alg\":\"RS256\",\"kid\":\"k54HQtDibyGcs9ZWVMfviHf-2
       qLcFUtpwY2rgxBk88M\"}";
13 String jws = generateJWS(header, payload);
14 httpServletRequest.setAttribute("request", jws);
```

```
1  private static String generateCodeChallenge(String randomString) {
2         byte[] hash = null;
3         try {
4                MessageDigest md = MessageDigest.getInstance("SHA-256");
5                hash = md.digest(randomString.getBytes(StandardCharsets.UTF_8))
                     ;
```

```
6         } catch (NoSuchAlgorithmException e) {
7                 e.printStackTrace();
8         }
9         String base64Url = Base64.getUrlEncoder().withoutPadding().
              encodeToString(hash);
10        return base64Url;
11  }
```

```
1   public String generatePayload(String nonce, String state, String
        code_challenge, JSONObject jsonObjectRequest) {
2         JSONObject idTokenClaims = new JSONObject();
3         idTokenClaims.put("https://attributes.spid.gov.it/familyName", new
              JSONObject().put("essential", true));
4         idTokenClaims.put("https://attributes.spid.gov.it/email", new
              JSONObject().put("essential", true));
5
6         Map<String, String> userinfoMap = new HashMap<>();
7         userinfoMap.put("PersonIdentifier", "https://attributes.spid.gov.it/
              fiscalNumber");
8         userinfoMap.put("FiscalNumber", "https://attributes.spid.gov.it/
              fiscalNumber");
9         userinfoMap.put("FirstName", "https://attributes.spid.gov.it/name");
10        userinfoMap.put("FamilyName", "https://attributes.spid.gov.it/
              familyName");
11        userinfoMap.put("DateOfBirth", "https://attributes.spid.gov.it/
              dateOfBirth");
12        userinfoMap.put("Email", "https://attributes.spid.gov.it/email");
13        userinfoMap.put("CurrentAddress", "https://attributes.spid.gov.it/
              address");
14        userinfoMap.put("Gender", "https://attributes.spid.gov.it/gender");
15        userinfoMap.put("PlaceOfBirth", "https://attributes.spid.gov.it/
              placeOfBirth");
16
17        JSONArray attrList = jsonObjectRequest.getJSONObject("
              authentication_request").getJSONArray("attribute_list");
18        JSONObject userinfoClaims = new JSONObject();
19        for (int i = 0; i < attrList.length(); i++) {
20            JSONObject attrObj = attrList.getJSONObject(i);
21            for (Entry<String, String> entry : userinfoMap.entrySet()) {
22                if (attrObj.getString("name").equals(entry.getKey())) {
23                    userinfoClaims.put(entry.getValue(), JSONObject.NULL);
24                }
25            }
26        }
27
28        JSONObject claims = new JSONObject();
29        claims.put("id_token", idTokenClaims);
30        claims.put("userinfo", userinfoClaims);
31        JSONArray aud = new JSONArray();
32        aud.put("http://trust-anchor.org:8000/oidc/op/");
33        aud.put("http://trust-anchor.org:8000/oidc/op/authorization");
34        JSONObject json = new JSONObject();
35        json.put("iss", "http://trust-anchor.org:8000/oidc/rp/");
36        json.put("scope", "openid");
```

```
37          json.put("redirect_uri", "http://eidas.org:8080/SpecificProxyService/
                OIDCServlet");
38          json.put("response_type", "code");
39          json.put("nonce", nonce);
40          json.put("state", state);
41          json.put("client_id", "http://trust-anchor.org:8000/oidc/rp/");
42          json.put("endpoint", "http://trust-anchor.org:8000/oidc/op/
                authorization");
43          json.put("acr_values", "https://www.spid.gov.it/SpidL2");
44          json.put("iat", 1678123154);
45          json.put("exp", 1678123214);
46          json.put("jti", "25d44402-e3d7-4204-8d5a-bdeff4167f14");
47          json.put("aud", aud);
48          json.put("claims", claims);
49          json.put("prompt", "consent login");
50          json.put("code_challenge", code_challenge);
51          json.put("code_challenge_method", "S256");
52
53          return json.toString();
54  }
```

```
1  public static String generateJWS(String header, String payload) {
2          String jws = base64UrlEncode(header.getBytes()) + "." +
               base64UrlEncode(payload.getBytes());
3          String signedJWS = base64UrlEncode(header.getBytes()) + "." +
               base64UrlEncode(payload.getBytes()) + "." + signJWS(jws);
4          return signedJWS;
5  }
```

```
1  private static String signJWS(String jwtUnsigned) {
2          String signatureBase64 = "";
3          try {
4              String privateKeyString = ...'';
5              privateKeyString = privateKeyString.replaceAll("-----BEGIN PRIVATE
                   KEY-----", "");
6              privateKeyString = privateKeyString.replaceAll("-----END PRIVATE
                   KEY-----", "");
7              privateKeyString = privateKeyString.replaceAll("\r\n", "");
8              privateKeyString = privateKeyString.replaceAll("\n", "");
9
10             // Decode the base64-encoded private key
11             byte[] privateKeyBytes = Base64.getDecoder().decode(
                   privateKeyString);
12
13             PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(
                   privateKeyBytes);
14             KeyFactory keyFactory = KeyFactory.getInstance("RSA");
15             PrivateKey privateKey = keyFactory.generatePrivate(keySpec);
16
17             Signature signature = Signature.getInstance("SHA256withRSA");
18             signature.initSign(privateKey);
19             signature.update(jwtUnsigned.getBytes());
```

```
20              byte[] signatureBytes = signature.sign();
21              signatureBase64 = Base64.getUrlEncoder().withoutPadding().
                    encodeToString(signatureBytes);
22          } catch (Exception e) {
23              e.printStackTrace();
24          }
25
26          return signatureBase64;
27  }
```

3.5.3 Nuovo servlet OIDC per lo Specific Proxy Service del nodo eIDAS

Una volta giunto all'authorization endpoint dell'OP, è richiesto allo user di inserire le credenziali personali per effettuare il login presso il fornitore di identità del proprio paese d'origine, in questo caso SPID OIDC. Successivamente viene presentata una pagina di consent per autorizzare gli attributi richiesti.

Con l'approvazione dell'utente, l'OP, procede quindi ad inviare una risposta alla redirect uri settata nella richiesta di autenticazione, e che corrisponde al nuovo OIDC servlet implementato nella componente dello Specific Proxy Service di eIDAS per gestire il seguente flusso OAuth 2.0 e OpenID Connect.

I parametri trasmessi sono in particolare lo stato (che l'RP deve verificare con quello inviato nella richiesta iniziale) e l'authorization code.

Il servlet OIDC avvia quindi il nuovo flusso interno OAuth 2.0 e OIDC, trasparente dal punto di vista dello user in quanto si svolge totalmente in background, non necessitando di ulteriori interventi da parte dello stesso. Viene prodotta una richiesta per raggiungere il token endpoint dell'OP, contenente come parametri la client assertion con i dati della richiesta, il code verifier, che l'OP deve controllare in corrispondenza al code challenge ricevuto precedentemente e ovviamente l'authorization code. Se tutto è conforme alle richiesta, l'OP, emette un access token e un ID token che fornisce in risposta al RP (il nuovo OIDC servlet in questo caso).

Infine, il servlet OIDC, utilizza l'access token ricevuto per invocare lo userinfo endpoint dell'OP, trasmettendolo come Bearer token nell'authorization header, e ottenento una risposta contenente gli attributi/claims richiesti sotto forma di JWT. La risposta dallo userinfo endpoint è in formato di JWT sotto forma di JWE, tocca quindi al servlet decriptare il contenuto e verificare la firma dell'OP prima di ottenere e ritenere valide le informazioni in esso contenute.

Metodi JAVA impiegati dal servlet OIDC per gestire il flusso di scambio messaggi con il provider OIDC:

```
1  protected void doGet(HttpServletRequest httpServletRequest,
       HttpServletResponse httpServletResponse)
2              throws ServletException, IOException {
3
4          //Receive here the authorization code and use it to get an access
               token from the token endpoint
5          String code = httpServletRequest.getParameter("code");
6          String state = httpServletRequest.getParameter("state");
7          String iss = httpServletRequest.getParameter("iss");
8
9          //Check the iss
10         if (!iss.equals("http%3A%2F%2Ftrust-anchor.org%3A8000%2Foidc%2Fop%2F")
               ) {
11             httpServletResponse.addHeader("iss error", "not matching " + iss);
12         }
```

```
13
14          //Check the state
15          ...
16          String requestState = contentState.toString();
17          if (!state.equals(requestState)) {
18              httpServletResponse.addHeader("state error", "not matching " +
                    state);
19          }
20
21          String access_token = "Bearer ";
22
23          try {
24              URL url = new URL("http://trust-anchor.org:8000/oidc/op/token/");
25              HttpURLConnection con = (HttpURLConnection) url.openConnection();
26              con.setRequestMethod("POST");
27
28              con.setRequestProperty("Content-Type", "application/x-www-form-
                    urlencoded");
29              String requestBody = constructRequestBody(code);
30
31              // Write the request body data to the connection's output stream
32              con.setDoOutput(true);
33              OutputStream outputStream = con.getOutputStream();
34              outputStream.write(requestBody.getBytes());
35              outputStream.flush();
36              outputStream.close();
37
38              int responseCode = con.getResponseCode();
39              String responseString = "";
40              if (responseCode == HttpURLConnection.HTTP_OK) {
41                  BufferedReader in = new BufferedReader(new InputStreamReader(
                        con.getInputStream()));
42                  String lineResponse;
43                  StringBuilder response = new StringBuilder();
44                  while ((lineResponse = in.readLine()) != null) {
45                      response.append(lineResponse);
46                  }
47                  in.close();
48                  responseString = response.toString();
49
50                  JSONObject tokenResponse = new JSONObject(response.toString());
51                  String token = tokenResponse.getString("access_token");
52                  access_token = access_token + token;
53                  httpServletResponse.addHeader("access_token", access_token);
54              } else {
55                  httpServletResponse.addHeader("token error", responseString);
56              }
57          } catch (Exception e) {
58              httpServletResponse.addHeader("error in token phase", e.toString())
                    ;
59          }
60
61          //Get the userinfo data by using the access token received
62          String userinfo = "";
63          try {
64              URL url = new URL("http://trust-anchor.org:8000/oidc/op/userinfo/")
                    ;
```

```
65              HttpURLConnection con = (HttpURLConnection) url.openConnection();
66              con.setRequestMethod("GET");
67              String token = access_token;
68
69              con.setRequestProperty("Authorization", token);
70
71              int responseCode = con.getResponseCode();
72              if (responseCode == HttpURLConnection.HTTP_OK) {
73                  BufferedReader in = new BufferedReader(new InputStreamReader(
                        con.getInputStream()));
74                  String lineResponse;
75                  StringBuilder response = new StringBuilder();
76                  while ((lineResponse = in.readLine()) != null) {
77                      response.append(lineResponse);
78                  }
79                  in.close();
80                  httpServletResponse.addHeader("userinfoJWE", response.toString
                        ());
81                  String decryptedJWE = decryptJwe(response.toString(), "/tmp/
                        rp_privateKey.txt");
82                  httpServletResponse.addHeader("decryptedJWE", decryptedJWE);
83                  verifySignature(decryptedJWE, "/tmp/op_publicKey.txt");
84                  String[] jwsParts = decryptedJWE.split("\\.");
85                  byte[] userinfoBytes = Base64.getDecoder().decode(jwsParts[1]);
86                  userinfo = new String (userinfoBytes);
87                  httpServletResponse.addHeader("userinfo", userinfo);
88              } else {
89                  httpServletResponse.addHeader("token expired", token);
90              }
91          } catch (Exception e) {
92              httpServletResponse.addHeader("error in userinfo phase", e.toString
                    ());
93          }
94
95          final String jSonResponseDecoded = "jSonResponseDecoded";
96          String SMSSPResponse = "";
97
98          ...
99
100         //Get the id of the original request
101         JSONObject jsonObjectRequest = new JSONObject(contentRequest.toString
                ());
102         String id = jsonObjectRequest.getJSONObject("authentication_request").
                getString("id");
103
104         JSONObject jsonObjectResponse = createResponse(id);
105         JSONObject responseObject = jsonObjectResponse.getJSONObject("response
                ");
106
107         //Create map from userinfo
108         try {
109             JSONObject userinfoObj = new JSONObject(userinfo);
110
111             Map<String, String> userinfoMap = new HashMap<>();
112             userinfoMap.put("PersonIdentifier", userinfoObj.getString("https://
                    attributes.spid.gov.it/fiscalNumber"));
```

```
113             userinfoMap.put("FiscalNumber", userinfoObj.getString("https://
                    attributes.spid.gov.it/fiscalNumber"));
114             userinfoMap.put("FirstName", userinfoObj.getString("https://
                    attributes.spid.gov.it/name"));
115             userinfoMap.put("FamilyName", userinfoObj.getString("https://
                    attributes.spid.gov.it/familyName"));
116             if (userinfoObj.has("https://attributes.spid.gov.it/dateOfBirth"))
117                 userinfoMap.put("DateOfBirth", userinfoObj.getString("https://
                        attributes.spid.gov.it/dateOfBirth"));
118             if (userinfoObj.has("https://attributes.spid.gov.it/email"))
119                 userinfoMap.put("Email", userinfoObj.getString("https://
                        attributes.spid.gov.it/email"));
120             if (userinfoObj.has("https://attributes.spid.gov.it/address"))
121                 userinfoMap.put("CurrentAddress", userinfoObj.getString("https
                        ://attributes.spid.gov.it/address"));
122             if (userinfoObj.has("https://attributes.spid.gov.it/gender"))
123                 userinfoMap.put("Gender", userinfoObj.getString("https://
                        attributes.spid.gov.it/gender"));
124             if (userinfoObj.has("https://attributes.spid.gov.it/placeOfBirth"))
125                 userinfoMap.put("PlaceOfBirth", userinfoObj.getString("https://
                        attributes.spid.gov.it/placeOfBirth"));
126
127             //Add userinfo to response model
128             JSONArray attrList = jsonObjectRequest.getJSONObject("
                    authentication_request").getJSONArray("attribute_list");
129             for (int i = 0; i < attrList.length(); i++) {
130                 JSONObject attrObj = attrList.getJSONObject(i);
131                 for (Entry<String, String> entry : userinfoMap.entrySet()) {
132                     if (attrObj.getString("name").equals(entry.getKey())) {
133                         JSONObject newAttribute = new JSONObject();
134                         newAttribute.put("type", "string_list");
135                         newAttribute.put("name", entry.getKey());
136                         JSONObject valuesObject = new JSONObject();
137                         valuesObject.put("value", entry.getValue());
138                         JSONArray valuesArray = new JSONArray();
139                         valuesArray.put(valuesObject);
140                         newAttribute.put("values", valuesArray);
141                         JSONArray respAttrList = responseObject.getJSONArray("
                                attribute_list");
142                         respAttrList.put(newAttribute);
143                     }
144                 }
145             }
146         } catch (Exception e) {
147             httpServletResponse.addHeader("userinfo exception", e.toString());
148         }
149
150         SMSSPResponse = Base64.getEncoder().encodeToString(jsonObjectResponse.
                toString().getBytes());
151
152         httpServletRequest.setAttribute("jSonResponseDecoded",
                jSonResponseDecoded);
153         httpServletRequest.setAttribute("SMSSPResponse", SMSSPResponse);
154         httpServletRequest.setAttribute("doNotmodifyTheResponse", "on");
155
156         RequestDispatcher dispatcher = httpServletRequest.getRequestDispatcher
                ("/oidcRedirect.jsp");
```

```
157         dispatcher.forward(httpServletRequest, httpServletResponse);
158 }
```

```
1  private String constructRequestBody(String code) {
2          ...
3          String code_verifier = contentCodeVerifier.toString();
4          String body = "client_id=" + "http://trust-anchor.org:8000/oidc/rp/" +
5                  "&client_assertion=" + client_assertion +
6                  "&client_assertion_type=" + "urn:ietf:params:oauth:client-
                       assertion-type:jwt-bearer" +
7                  "&code=" + code +
8                  "&code_verifier=" + code_verifier +
9                  "&grant_type=authorization_code";
10
11         return body;
12 }
```

```
1  private static String decryptJwe(String jwe, String privateKeyFilePath) throws
       Exception {
2          String[] jweParts = jwe.split("\\.");
3          if (jweParts.length != 5) {
4              throw new IllegalArgumentException("Invalid JWE format");
5          }
6
7          byte[] encryptedKey = Base64.getUrlDecoder().decode(jweParts[1]);
8          byte[] iv = Base64.getUrlDecoder().decode(jweParts[2]);
9          byte[] cipherText = Base64.getUrlDecoder().decode(jweParts[3]);
10         byte[] tag = Base64.getUrlDecoder().decode(jweParts[4]);
11
12         byte[] decryptedKeyBytes = decryptKey(encryptedKey, privateKeyFilePath
               );
13
14         byte[] truncatedKeyBytes = Arrays.copyOfRange(decryptedKeyBytes, 32,
               64);
15         SecretKey decryptionKey = new SecretKeySpec(truncatedKeyBytes, "AES");
16
17         Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
18         IvParameterSpec ivSpec = new IvParameterSpec(iv);
19         cipher.init(Cipher.DECRYPT_MODE, decryptionKey, ivSpec);
20
21         byte[] decrypted = cipher.doFinal(cipherText);
22         String result = new String(decrypted);
23         result = result.replace("\"", "");
24
25         return result;
26 }
```

```
1  private static byte[] decryptKey(byte[] encryptedKey, String
       privateKeyFilePath) throws Exception {
```

```
2         String privateKeyString = ...'';
3         privateKeyString = privateKeyString.replaceAll("-----BEGIN PRIVATE KEY
              -----", "");
4         privateKeyString = privateKeyString.replaceAll("-----END PRIVATE KEY
              -----", "");
5         privateKeyString = privateKeyString.replaceAll("\r\n", "");
6         privateKeyString = privateKeyString.replaceAll("\n", "");
7         // Decode the base64-encoded private key
8         byte[] privateKeyBytes = Base64.getDecoder().decode(privateKeyString);
9         PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(privateKeyBytes)
              ;
10        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
11        PrivateKey privateKey = keyFactory.generatePrivate(keySpec);
12
13        Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA1AndMGF1Padding
              ");
14        cipher.init(Cipher.DECRYPT_MODE, privateKey);
15        return cipher.doFinal(encryptedKey);
16    }
```

```
1   private static void verifySignature(String jws, String publicKeyFilePath)
        throws Exception {
2
3         String publicKeyString = ...''
4         publicKeyString = publicKeyString.replaceAll("-----BEGIN PUBLIC KEY
              -----", "");
5         publicKeyString = publicKeyString.replaceAll("-----END PUBLIC KEY
              -----", "");
6         publicKeyString = publicKeyString.replaceAll("\r\n", "");
7         publicKeyString = publicKeyString.replaceAll("\n", "");
8         // Decode the base64-encoded private key
9         byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyString);
10
11        X509EncodedKeySpec keySpec = new X509EncodedKeySpec(publicKeyBytes);
12        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
13        PublicKey publicKey = keyFactory.generatePublic(keySpec);
14
15        // Split the JWS into its three parts
16        String[] parts = jws.split("\\.");
17        if (parts.length != 3) {
18            throw new IllegalArgumentException("Invalid JWS format");
19        }
20
21        // Decode the signature from Base64
22        byte[] signature = Base64.getUrlDecoder().decode(parts[2]);
23
24        // Verify the signature using the public key
25        Signature verifier = Signature.getInstance("SHA256withRSA");
26        verifier.initVerify(publicKey);
27        verifier.update((parts[0] + "." + parts[1]).getBytes());
28
29        if (!verifier.verify(signature)) {
30            throw new Exception("Signature not verified");
31        }
32    }
```

3.5.4 Authentication response da SPID a eIDAS

Gli attributi ricevuti vengono converiti nella forma, con un ulteriore mapping, in un formato che il sistema di nodi eIDAS può comprendere, simile a quanto mostrato nell'esempio. Dal servlet OIDC la risposta viene quindi traferita al servlet ancora nello Specif Proxy Service che si occupa di ricevere la risposta dall'IdP per trasformarla in SAML response da resistituire al framework eIDAS con i relativi nodi e quindi indietro al eIDAS Connector del paese A e al SP che ha inviato la richiesta inziale.

```
{
   "response" : {
      "attribute_list" : [ {
         "type" : "string_list",
         "name" : "LegalName",
         "values" : [ {
            "value" : "Current Legal Name"
         } ]
      }, [...] {
         "type" : "string_list",
         "name" : "FamilyName",
         "values" : [ {
            "value" : "Garcia"
         } ]
      } [...] ],
      "authentication_context_class" : "A",
      "client_Ip_Address" : "192.168.64.1",
      "created_on" : "2023-02-13T00:55:20.499+01:00",
      "id" : "03a7759e-4df1-4ec4-8c22-0e136400c2aa",
      "inresponse_to" : " b14ffd42-6e34-4b46-8caf-268bc0d8fcd7",
      "issuer" : "DEMO-IDP",
      "name_id" : "unspecified",
      "status" : {
         "status_code" : "success"
      },
      "subject" : "0123456",
      "version" : "1"
   }
}
```

Metodo per la traduzione della risposta ottenuta sul servlet OIDC in un formato accettato da eIDAS:

```
1  private JSONObject createResponse(String id) {
2          JSONArray attributeList = new JSONArray();
3
4          // Create status object
5          JSONObject status = new JSONObject()
6                  .put("status_code", "success");
7
8          // Create response object
9          JSONObject response = new JSONObject()
10                 .put("response", new JSONObject()
11                         .put("attribute_list", attributeList)
12                         .put("authentication_context_class", "A")
```

```
13                          .put("client_Ip_Address", "127.0.0.1")
14                          .put("created_on", LocalDateTime.now().atOffset(
                                ZoneOffset.of("+01:00")).format(DateTimeFormatter.
                                ISO_OFFSET_DATE_TIME))
15                          .put("id", UUID.randomUUID().toString())
16                          .put("inresponse_to", id)
17                          .put("issuer", "DEMO-IDP")
18                          .put("name_id", "unspecified")
19                          .put("status", status)
20                          .put("subject", "0123456")
21                          .put("version", "1"));
22
23          return response;
24  }
```

Per questioni di tracciabilità e necessario l'accorgimento di inserire, nel campo *inresponse_to* della risposta per eIDAS, il valore del campo *id* della richiesta originale elaborata dallo Specific Proxy Service prima di essere customizzata per raggiungere l'OP. In questo modo è possibile per eIDAS tenere traccia del flusso e associare la richiesta alla relativa risposta ottenuta.

3.6 Testing della soluzione

Dopo aver installato e configurato opportunamente i due sistemi relativi ad eIDAS, con il web server Tomcat ed i relativi file WAR deploiati ed avviati, ed a SPID OIDC, con docker-compose, e dopo aver compiuto ed apportato le modifiche descritte nei paragrafi precedenti, è possibile procedere ad un test per simulare un'interazione d'uso comune che metta in evidenza l'integrazione di OIDC nel framework di eIDAS.

1. Al path http://eidas.org:8080/SP è possibile raggiungere il servlet di Tomcat che ospita un SP di demo per eIDAS. L'SP in questione è assimilabile ad un SP a cui intende accedere un utente in un paese europeo diverso da quello di origine, al fine di ottenere i relativi servizi, ad esempio per il pagamento di tasse statali.

2. Il form della pagina su cui si atterra presenta la possibilità di scegliere il paese di origine dell'utente e quello del SP simulato. Per questioni di demo, in entrambi i casi, il valore da inserire dovrà essere *CA* (Contry A). In realtà, il SP, come detto, dovrebbe trovarsi in un country A mentre il cittadino, user, dovrebbe avere nazionalità in un paese diverso, come ad esempio country B, tuttavia, per motivi di demo, iin questo caso, il valore CA, rappresenta soltanto un riferimento al web server con cui è stato deploiato il sistema eIDAS, in questo caso Tomcat porta 8080, che espone le relative componenti,q uindi Specific Connector e Specific Proxy Service di eIDAS.

3. Sulla pagina del demo SP è possibile configurare anche gli attributi che il SP intende richiedere all'IdP per autenticare ed ottenere informazioni circa l'utente richiedente. Obbligatori devono essere qui, sempre, gli attributi *PersonIdentifier*, *DateOfBirth*, *FamilyName*, *LegalPersonIdentifier*, *LegalName*, necessari per una corretta minima autenticazione eIDAS. Se, infatti, lo user che viene successivamente autenticato con l'IdP SPID OIDC, non possiede qualcuno di questi attributi, la richiesta eIDAS, terminerà sempre con esito negativo nel momento di valutazione da parte del demo SP.

4. Dopo aver cliccato su *Submit*, il framework di eIDAS, guida lo user verso l'endpoint espone il servizio Specific Proxy Service di eIDAS, passando (in modo del tutto trasparente all'utente, per le componenti di Specific Connector ed eIDAS Node nazionale). Qui viene richiesto al cittadino richiedente di autorizzare una prima volta gli attributi che il SP intende richiedere (configurati nella pagina precedente per scopi di demo), ed eventualmente, cliccando su *Next*, permettere al SP di ottenerne altri optional oltre quelli già indicati come obbliagatori.

5. Si atterà quindi sulla pagina del consenso e poi si viene reindirizzati alla pagina di SPID per il login (purtroppo il sistema di demo indicato per SPID OIDC supporta solo l'autenticazione con livello spidL2). La pagina di spid in questione rappresenta l'authorization server del sistema SPID con OIDC, al quale è già giunta la richiesta con gli attributi da parte dello Specific Proxy Service, e che adesso richiede appunto all'utente di autenticarsi per dimostrare la propria identità.

6. Il login può essere effettuato con gli user di esempio user:oidcuser o admin:oidcadmin forniti di default con il sistema SPID, oppure con altre utenze in precedenza configurate sul file example.json del sistema SPID. Come indicato prima, i due utenti di default, non possiedono l'attributo obbligatorio *dateOfBirth*, quindi, a meno di modifiche per ovviare a tale problematica, la richiesta terminerà in questi casi, infine, con un errore.

7. Il sistema SPID, dopo il login positivo, chiede all'utente di fornire il consenso per utilizzare gli attributi SPID indicati a video. Questa parte di interazione con l'utente, permette di dare via al flusso proprio di OAuth 2.0 con OIDC, che mette in funzione il nuovo servlet OIDC creato nella parte di MS Specific per il Proxy Service di eIDAS, che effettua quindi, in modo del tutto invisibile all'utente, tutte le operazioni necessarie, di scambio authorization code per access token e poi richiesta allo userinfo endpoint dell'IdP per ottenere gli attributi richiesti, con annesse le verifiche di firme, encryption ed decryption di contenuti.

8. Lo user verrà a questo punto indirizzato nuovamente verso lo Specific Proxy Service, nella sezione in cui vengono visualizzati gli attributi (adesso popolati) ottenuti da SPID OIDC dopo i precedenti consensi, con ulteriore richiesta di verifica e conferma degli stessi.

9. Infine, in modo ancora trasparente all'utente, il giro di test percorrerà indietro gli step tornando all'eIDAS Node e quindi allo Specific Connector ed alla pagina di riepilogo del demo SP di eIDAS, in cui gli attributi popolati vengono visualizzati dimostrando la possibilità da parte del SP di fruire degli stessi per permettere allo user richidente di accedere ai relativi servizi.

Capitolo 4

Risultati e considerazioni

4.1 Comparazione performance con SAML

OAuth 2.0 con OIDC è un protocollo moderno, realizzato in modo specifico per autorizzazione ed autenticazione su sistemi web-based. Considerando la difficoltà implementativa, OIDC, che è realizzato come layer aggiuntivo su OAuth 2.0 per permettere anche l'autenticazione, risulta più facile da implementare rispetto a SAML, designato originariamente per la gestione di identità federate.

L'utlizzo di OIDC risulta quindi in un meccanismo più leggero e flessibile di SAML, mediante soprattutto l'utilizzo di JSON Web Token (JWT) per lo scambio di token o altre informazioni rilevanti, in modo sicuramente più compatto e rapido rispetto a quanto permesso dalle assertion SAML basate su XML.

Lo scenario indicato rappresenta quindi un'applicazione pratica in cui OIDC permette, tramite un utilizzo efficiente di scambio di token, la riduzione del sovraccarico computazionale necessario per il processo di autorizzazione ed autenticazione. OIDC può essere reso, inoltre, anche più rapido grazie all'adozione possibile di meccanismi di caching e altre tecniche di ottimizzazione per ridurre la latenza e migliorare i tempi di risposta.

Considerando in analisi i possibili volumi, elevati, previsti da un'eventuale integrazione tra EIDAS e SPID con OIDC, che rappresenterebbe il meccanismo ufficila per la gestione di identità digitali attraverso i confini europei, il beneficio apportato sarebbe essenziale per questo tipo di transazioni in cui ogni millisecondo può far la differenza.

Un ulteriore aspetto da considerare è l'implementazione specifica del protocollo. Risulta infatti necessario seguire le linee guida di iGov per permettere un'applicazione sicura ed efficiente di OIDC, che potrebbe così contrapporsi alla complessità di implementazione, sviluppo, fornitura e mantenimento di un sistema complesso come quello basato su SAML.

Infine, la possibilità di utilizzare delle sessioni a lungo termine, con l'utilizzo di refresh token, permesso da OIDC, crea una forte modalità di miglioramento delle performance offerte dall'adozione di questo protocollo. Gli user potrebbero trarne beneficio e, in molti casi in cui questo meccanismo risulta possibile, potrebbe ridurre al minimo le tempistiche ed i passaggi necessari per l'autenticazione e l'autorizzazione delle risorse presso i propri IdP nazionali.

4.2 Analisi sulla user experience

La maggior parte di passaggi richiesti dall'integrazione di OIDC nel sistema EIDAS, risulta del tutto trasparente all'utente. Tuttavia, anche da questo punto di vista, è possibile notare dei miglioramenti per quanto riguarda un possibile utilizzo più user-friendly, a cui l'utente è per l'apppunto già abituato dal punto di vista di interazioni nell'utilizzo di autenticazione e autorizzazione all'interno dei propri confini nazionali.

Il beneficio aggiuntivo dato da questa integrazione risulta quindi in possibilità di utilizzo di sessioni revocabili a lungo termine, che semplificano l'interazione da parte dello user e gli permettono anche di aver un controllo più dettagliato e completo sull'utlizzo e la condivisione delle informazioni e delle risorse personali, fornendo anche la possibilità di decidere se e como revocare eventuali permessi.

Perdipiù, la compatibilità che OAuth 2.0 con OIDC offrono, in combinazione con dispositivi mobili per l'utilizzo di app mobile e dispositivi IoT, permette sicuramente una maggiore facilità ed interoperabilità wireless ed in movimento per molti user che, come rilevato negli ultimi anni, sono sempre più indirizzati, o quasi esclusivamente indirizzati, all'utilizzo di tecnologie mobili, differentemente da quelle più "fisse" con le quali SAML aveva una maggiore compatibilità.

4.3 Considerazioni di sicurezza e privacy

OIDC permetto un meccanismo indubbiamente sicuro per l'autenticazione e l'autorizzazione, riducendo al minimo il rischio di vulnerabilità associate se correttamente implementato e mantenuto. Ciò risulta in particolare vero se adottato con le linee guida che lo potenziano in tal senso, offerte dallo standard dato da iGov.

Un altro aspetto fondamentale che rafforza la sicurezza di questo nuovo sistema, è da attribuire alla forte possibilità di integrazione con altri strumenti di sicurezza, quali as esempio firewall o altri sistemi per l'individuzione di attività malevole, con i quali è facilmente interoperare considerando che il sistema fa utilizzo di API.

Per quanto riguarda la privacy, l'utilizzo di token basati su JWE, in combinazione con il protocollo HTTPS per le tutte le transazioni e lo scambio di messaggi, permette un alto livello di protezione dei dati personali e delle informazioni scambiate tra le entità coinvolte, con l'aggiunta di meccanismi con il perfect-forward-secrecy che giocano un ruolo fondamentale anche in caso di successiva compromissione.

Infine, con l'adozione di OIDC, lo user può avere un controllo più preciso riguardo il rilascio delle proprie informazioni: i dati, infatti, sono condivisi esclusivamente quanto necessario e sotto attenta autorizzazione da parte di chi li possiede, con l'aggiunta, come detto prima, della possibilità di revocare eventuali sessioni attive (revocando i relativi access token per l'ottenimento degli attributi degli user), fermando, anche prima del tempo necessario eventualmente, l'accesso alle proprie risorse per conto di terzi, come ad esempio un SP a cui si è fatta inizialmente, ma erroneamente richiesta.

4.4 Lavori futuri e suggerimenti

Durante una prima fase di integrazione, alcuni degli IdP nazionali, continueranno ad utilizzare standard come XML e le assertion basate su SAML, mentre altri possederanno già la tecnologia necessaria per gestire i flussi OAuth 2.0 con OIDC. Sarebbe quindi sicuramente necessario prevedere un primo periodo, di transizione tecnologica, iin cui esiste la possibilità da parte dei nodi EIDAS, di scegliere il formato e le modalità da adottare in funzione dell'IdP che lo user ha selezionato.

Infine, il passaggio tecnologico che vede il favorimento di protocolli moderni come OIDC rispetto a SAML, potrebbe essere esteso, oltre che all'integrazione tra EIDAS e i vari IdP nazionali, anche all'interno del framework di nodi EIDAS stesso. Infatti, internamente, la tecnologia adottata per lo scambio di informazione di autorizzazione e autenticazione tra i vari componenti di EIDAS, tra cui i Connector, i Proxy Service e tutti i nodi interni conivolti in un flusso standard, si basa ancora su SAML 2.0.

Un aggiornamento in tal senso potrebbe sicuramente migliorare in modo esteso e generale, le performance, la scalabilità e anche il mantenimento dei sistemi che gestiscono l'infrastruttura europea di identità digitale.

Capitolo 5

Conclusione

5.1 Riassunto dello studio

L'implementazione proposta può essere considerata come una delle possibili soluzioni attuabili per permettere l'integrazione tra il sistema eIDAS ed un identity provider, che nel caso specifico era SPID, dotato di tecnologia e protocolli OAuth 2.0 con OpenID Connect. Le modifiche apportate risultano circoscritte, come precedentemente indicato, alla componente specifica per stato membro del Proxy Service in un nodo eIDAS, permettendo un'eventuale integrazione puntuale e mirata.

L'elemento principale è sicuramente da individuarsi nel nuovo servlet OIDC che permette alla componente Specific Proxy Service di funzionare come una relying party secondo la terminologia del flusso OAuth 2.0. In modo trasparente all'utente, infatti, è proprio attraverso questo servlet che ha veramente luogo l'integrazione con l'identity provider che utilizza OIDC.

Non sono inoltre necessari cambiamenti dal punto di vista dell'identity provider, utilizzando infatti la versione di demo di SPID con OIDC prodotta da AGID secondo le linee guida iGov, i metodi e gli schemi del sistema si sono mantenuti come da default. Questo è il comportamento che ci si dovrebbe infatti aspettare da un integrazione che è maggiormente rivolta all'aggiornamento tecnologico del framework di eIDAS, lasciando libertà all'IdP, che non deve essere condizionato dagli schemi di eIDAS, di gestire internamente i propri sistemi, con la sola prerogativa di essere compliant alle direttive, ad esempio quelle di iGov, che rendono sicuro e danno uno standard di funzionamento al meccanismo basato sui protocolli OAuth 2.0 e OIDC.

I sistemi di cache, già implementati nell'infrastruttura di funzionamento tecnico per eIDAS, potrebbero essere utilizzate per le sessioni a lungo termine proprie di OAuth 2.0 e OIDC in cui può essere bypassato il passaggio di authentication request se lo Specific Proxy Service dovesse risultare in possesso di un access token o un refresh token valido per accedere agli attributi dello user presso l'identity provider.

RESTful api e JSON sono le tecnologie sulle quali la nuova soluzione fonda gran parte del funzionamento. In particolare, le richieste e le relative risposte scambiate tra lo Specific Proxy Service e gli endpoint authorization server, token e userinfo dell'identity provider, avvengono tutte per mezzo di api che, per l'appunto, espongono le risorse, con comunicazioni in formato JSON anche nella creazione di parti codificate e firmate, come visto con l'impiego di JWE e JWS, facilmente gestibili dai linguaggi di programmazione e tecnologie moderne e da dispositivi caratterizzati da un uso di tipo mobile.

Per quanto riguarda gli aspetti di sicurezza, infine, con un'implementazione corretta, che segua pedissequamente le linee guida ufficiali, è possibile garantire un livello di protezione dei sistemi, delle identità coinvolte nell'autenticazione e delle risorse esposte, adeguato alla richieste della nostra era interconnessa in cui è piuttosto semplice esporre, ed esporsi, a minacce quali ad esempio, in primo luogo, il furto di identità in ambito digitale con ripercussioni online e spesso, anche, offline. Questo fine è raggiunto dall'integrazine di OAuth 2.0 con OIDC mediante l'uso di TLS sempre presente per tutte le transazioni tra entità, l'utilizzo di certificati e chiavi pubbliche

e private con meccanismi di perfect-forward-secrecy per le chiavi simmetriche, che permettono nel complesso integrità, confidenzialità e privacy del dato, anche a medio-lungo termine, con firme e codifiche dei dati ed inoltre potenziato dall'impiego di codici di autorizzazione e token per la gestione delle risorse, senza che la sessione instaurata tra le entità coinvolte sia direttamente continuamente attiva.

5.2 Implicazioni per l'infrastruttura pan-Europea di identità digitale

Al fine di garantire il funzionamento del meccanismo previsto da OAuth 2.0 con OIDC, il sistema eIDAS deve essere registrato in una federazione, in cui partecipa come relying party, comunemente denominato client, insieme all'identity provider di riferimento, che in questo caso possiamo assumere come SPID. All'interno di questa federazione, lo Specific Proxy Service in particolare, di eIDAS, deve fornire i metadati necessari per l'individuazione degli endpoint utili al flusso di OAuth 2.0 e le chiavi pubbliche con le quali sarà poi possibile per l'identity provider procedere alla codifica dei dati.

La *redirect_uri* anche conosciuta come callback uri, dovrà quindi essere esposta pubblicamente, secondo il principio contrario alla Security Through Obscurity, e puntare al nuovo servlet OIDC implementato, che l'identity provider di turno utilizzerà per ridirezionare l'authorization code proprio del flusso OAuth 2.0 con OIDC, dopo il login riuscito correttamente da parte dello user. Dall'altro lato, il servlet OIDC di eIDAS, dovrà anche essere in grado di reperire le informazioni relative ai metadata del corrispettivo, o dei corrispettivi, IdP, come da registro della federazione (ad esempio come indicato dal registro di SPID), per permettere di raggiungere i vari endpoint quali authrorization server, token e userinfo e utilizzare le chiavi pubbliche dell'IdP per decodificare i dati ricevuti e verificare le relative firme digitali.

Il più grosso impatto dal punto di vista di eIDAS, se così si può definire, sarà dunque quello di esporre e rendere attivo il nuovo servlet OIDC che gestisca quasi interamente, a meno della parte di invio iniziale dell'authentication request che dovrà ancora essere svolta da un altro servlet dello Specific Proxy Service (già presente in eIDAS) e attivo in fase pressochè iniziale quando per la prima volta viene richiesto allo user di autorizzare la richiesta per determinati attributi. Tale esposizione dovrà essere resa pubblica per permettere la raggiungibilità del servlet dal fornitore di identità, tuttavia, non sarà necessario curarne altri aspetti in quanto la nuova implementazione sarà del tutto trasparente e non avrà mai dirette interazioni con lo user finale. In tal senso, sarebbe ad esempio attuabile, un sistema di limitazione all'accesso di tale nuovo servlet di eIDAS solo per gli identity provider precedentemente approvati e già registrati nella federazione.

5.3 Considerazioni finali e prospettive future

Il Regolamento europeo per l'identificazione elettronica e servizi fiduciari per le transazioni elettroniche nel mercato interno (altro modo per indicare eIDAS) è un regolamento dell'Unione europea, ufficiale (UE) n. 910/2014, che riguarda l'identificazione elettronica e i servizi fiduciari per le transazioni elettroniche nel Mercato europeo comune. Come si può evincere dalla riga precedente, esso è entrato in vigore nel Luglio 2014, ed oggi, alla soglia di quasi un decennio di attività, si è mantenuto pressochè tecnologicamente inviariato rispetto alla prima edizione.

Lo standard utilizzato è sempre stato SAML (seppur con modifiche interne per la gestione delle transazioni, delle comunicazioni tra i vari nodi e i meccanismi di cache per evitare parzialmente di esporre le informazioni e diminuire i tempi di risposta) anche in fase di comunicazione con i vari IdP nazionali e risulta quindi sicuramente necessaria la predisposizione oggi ad accogliere nuove tecnologie, come OAuth 2.0 e OpenID Connect, che stanno riscontrando il favore dei sistemi di autenticazione digitale in tutto il mondo.

In un prima fase risulterà sicuramente necessario provvedere ad una gestione ambivalente dei due flussi, SAML e OAuth 2.0 con OIDC, e farli coesistere permettendo ad eIDAS di scegliere

la modalità in funzione dell'identity provider selezionato dallo user nel caso specifico. Questo porterà ad una progressiva diminuzione degli IdP che utilizzano ancora lo standard SAML, a volte si tratta anche di un solo IdP nazionale per alcuni paesi europei, fino a quando, infine, il nuovo standard verrà reso mandatorio, auspicabilmente con confini e linee guida ben precisi e comuni a tutti gli stati membri coinvolti.

Altre possibilità, attualmente in fase di studio, sono relative all'utilizzo di eIDAS come una sorta di proxy che non gestisce dati al suo interno, nè richieste, nè risposte, ma è necessario solo a mettere in comunicazione service provider ed identity provider di paesi diversi, in funzione della locazione e della provenienza dello user che necessita disporre della sua identità digitale. Questo sistema escluderebbe quasi del tutto la necessità di un'integrazione con nuove tecnologie, ed in generale di molte delle logiche attualmente attive, nel sistema eIDAS, ma porterebbe in primo la piano la necessità di individuare flussi e funzionamenti del tutto nuovi e diversi da quelli attualmente in vigore, senza tralasciare il fondamentale aspetto della sicurezza e delle regolazioni legislative che creano spesso, paradossalmente, un punto di innesto, seppur apprentemente laborioso e complicato, tra le varie giurisdizioni locali.

In aggiunta, secondo uno studio già affrontato ed ampiamente discusso nel paper "On Enabling Additional Natural Person and Domain-Specific Attributes in the eIDAS Network" [1], potrebbe essere implementato un meccanismo aggiuntivo che preveda una maggiore completezza delle informazioni nel momento in cui un service provider abbia la necessità di richiedere dati particolari relativi ad uno user. In questo caso, infatti, il flusso tra eIDAS e l'identity provider, seguirebbe lo stesso corso già indicato, ma in aggiunta, prima di concludere la transazione presso il Proxy Service del nodo eIDAS, sarebbe possibile inteorrogare un aggiuntivo "Attribute Server", esterno all'IdP, in grado di fornire gli attributi particolari aggiuntivi richiesi di cui il fornitore di identità non disponeva.

Per concludere, come da notizie delle ultime settimane, è probabile che in Italia, il sistema SPID, adottato in questo progetto di tesi come modello di IdP con OIDC per l'integrazione con eIDAS, potrebbe essere interrotto in favore di un sistema di identità digitale centralizzato nazionale basato esclusivamente sull'utilizzo di carta di identità elettronica, CIE. Come accennato, tuttavia, l'integrazione discussa e la soluzione proposta con i relativi vantaggi, manterrebbero comunque la loro validità in quanto l'implementazione è relativa al protocollo OAuth 2.0 con OIDC all'interno dell'infrastruttura eIDAS, indipendetemente dal tipo di fornitore di identità e anche, per completezza, del paese scelto in cui applicare la modifica alla parte specifica del Proxy Service tra tutte le nazioni in cui è in vigore la regolamentazione e quindi in possesso di un nodo eIDAS nazionale.

[1] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9548914

Appendice A

Programmer manual

A.1 Ambiente virtuale

A.1.1 Macchina virtuale

1. Installazione software di virtualizzazione VirtualBox v7.0 (https://www.virtualbox.org/wiki/Downloads).

2. Creazione macchina virtuale da ISO, con sistema operativo Linux Ubuntu v22.04. Risorse minime: 2GB di RAM.

A.1.2 JAVA

1. Installazione JAVA JDK 11 per maggiore compatibilità col software di eIDAS.

   ```
   sudo apt-get install openjdk-11-jdk
   ```

2. Modifica variabili d'ambiente con l'aggiunta del path per la home di JAVA.

   ```
   1 sudo vim /etc/environment
   2 -> JAVA_HOME="/usr/lib/jvm/java-11-openjdk-amd64"
   3 source /etc/environment # per rendere effettivo il path
   ```

A.1.3 Libreria Bouncy Castle Provider

1. Download libreria Bouncy Castle Provider per garantire le operazioni di sign/verify/encrypt/decrypt data (https://www.bouncycastle.org/download/bcprov-jdk15to18-172.jar).

2. Copiare al path /home/user/Downloads.

3. Modifica del file java.security abilitanto la libreria Bouncy Castle Provider (il valore di N è scelto in base al numero di provider già esistenti nel file e segue un ordine crescente).

   ```
   1 sudo vim $JAVA_HOME/conf/security/java.security
   2 -> security.provider.N=org.bouncycastle.jce.provider.BouncyCastleProvider
   ```

A.2 Installazione di eIDAS 2.6

A.2.1 Download codice eIDAS 2.6

1. Da browser: https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eIDAS-Node+version+2.6.
2. Copiare al path /home/user/Desktop e unzip.
3. Unzip anche della cartella interna EIDAS-Sources-2.6.0.zip e EIDAS-Binaries-Tomcat-2.6.0.zip.
4. Copia keystore e tomcat della cartella unzippata da EIDAS-Binaries-Tomcat-2.6.0.zip nella cartella EIDAS-Sources-2.6.0/EIDAS-Config/.

A.2.2 Pre-configurazione Tomcat

1. Download web server Tomcat e creazione user e folder.

```
1 sudo useradd -m -U -d /opt/tomcat -s /bin/false tomcat
2 VERSION=9.0.73 # (verificare ultima versione disponibile)
3 wget https://www-eu.apache.org/dist/tomcat/tomcat-9/v${VERSION}/bin/
      apache-tomcat-${VERSION}.tar.gz -P /tmp
4 sudo tar -xf /tmp/apache-tomcat-${VERSION}.tar.gz -C /opt/tomcat/
5 sudo ln -s /opt/tomcat/apache-tomcat-${VERSION} /opt/tomcat/latest
6 sudo chown -R tomcat: /opt/tomcat
7 sudo sh -c chmod +x /opt/tomcat/latest/bin/*.sh'
```

2. Copia libreria Bouncy Castle Provider per utilizzo Tomcat.

```
1 sudo cp /home/user/Downloads/bcprov-jdk15to18-172.jar /opt/tomcat/latest/
      bin
2 sudo chown tomcat.tomcat /opt/tomcat/latest/bin/bcprov-jdk15to18-172.jar
```

3. Copia configurazioni eIDAS per utilizzo Tomcat.

```
1 sudo cp -R /home/user/Desktop/eIDAS-node-2.6.0/EIDAS-Sources-2.6.0 /opt/
      tomcat/latest/work
2 sudo chown -R /opt/tomcat/latest/work/EIDAS-Sources-2.6.0
```

4. Creazione Apache Ignite working directory (cache condivisa per eIDAS).

```
1 sudo mkdir /ignite/work
2 sudo chmod 777 -R /ignite
```

5. Creazione script di configurazione per il servizio Tomcat con l'aggiunta delle variabili per le cartelle di configurazione di eIDAS e la libreria Bouncy Castle Provider).

```
sudo vim /etc/systemd/system/tomcat.service # di seguito il contenuto del
     file
```

```
1 [Unit]
2 Description=Tomcat 9 servlet container
3 After=network.target
4
5 [Service]
6 Type=forking
7
8 User=tomcat
9 Group=tomcat
```

```
10
11 Environment="JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64"
12 Environment="JAVA_OPTS=-Djava.security.egd=file:///dev/urandom -Djava.awt
       .headless=true -DEIDAS_CONFIG_REPOSITORY=/opt/tomcat/latest/work/
       EIDAS-Sources-2.6.0/EIDAS-Config/tomcat/ -
       DSPECIFIC_CONNECTOR_CONFIG_REPOSITORY=/opt/tomcat/latest/work/EIDAS-
       Sources-2.6.0/EIDAS-Config/tomcat/specificConnector/ -
       DSPECIFIC_PROXY_SERVICE_CONFIG_REPOSITORY=/opt/tomcat/latest/work/
       EIDAS-Sources-2.6.0/EIDAS-Config/tomcat/specificProxyService/ -
       DSP_CONFIG_REPOSITORY=/opt/tomcat/latest/work/EIDAS-Sources-2.6.0/
       EIDAS-Config/tomcat/sp/ -DIDP_CONFIG_REPOSITORY=/opt/tomcat/latest/
       work/EIDAS-Sources-2.6.0/EIDAS-Config/tomcat/idp/ --module-path /opt/
       tomcat/latest/bin/bcprov-jdk15to18-172.jar --add-modules org.
       bouncycastle.provider"
13
14 Environment="CATALINA_BASE=/opt/tomcat/latest"
15 Environment="CATALINA_HOME=/opt/tomcat/latest"
16 Environment="CATALINA_PID=/opt/tomcat/latest/temp/tomcat.pid"
17 Environment="CATALINA_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC"
18
19 ExecStart=/opt/tomcat/latest/bin/startup.sh
20 ExecStop=/opt/tomcat/latest/bin/shutdown.sh
21
22 [Install]
23 WantedBy=multi-user.target
```

A.2.3 Attivizatione servizio Tomcat (port 8080)

1. Caricamento script di configurazione e abilitazione all'avvio del sistema.

```
1 sudo systemctl daemon-reload
2 sudo systemctl enable --now tomcat
3 sudo systemctl status tomcat
```

2. Creazione user admin.

```
sudo vim /opt/tomcat/latest/conf/tomcat-users.xml
```

```
1 <tomcat-users>
2         <!--
3         Comments
4         -->
5         <role rolename="admin-gui"/>
6         <role rolename="manager-gui"/>
7         <user username="admin" password="tomcat" roles="admin-gui,manager-
              gui"/>
8 </tomcat-users>
```

```
sudo vim /opt/tomcat/latest/webapps/manager/META-INF/context.xml
```

```
1 <Context antiResourceLocking="false" privileged="true" >
2         <!--
3         <Valve className="org.apache.catalina.valves.RemoteAddrValve"
4                allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
5         -->
6 </Context>
```

```
sudo vim /opt/tomcat/latest/webapps/host-manager/META-INF/context.xml
```

```
<Context antiResourceLocking="false" privileged="true" >
        <!--
        <Valve className="org.apache.catalina.valves.RemoteAddrValve"
               allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
        -->
</Context>
```

A.2.4 Modifica file hosts

1. Modifica file hosts per raggiungere il sistema eIDAS al dominio eidas.org.

```
vi /etc/hosts
-> 127.0.0.1 localhost eidas.org
```

A.2.5 Deploy su Tomcat

1. Spostare i file EidasNode.war, SP.war, IdP.war, SpecificConnector.war, SpecificProxyService.war dalla cartella /home/user/Desktop/eIDAS-node-2.6.0 nella cartella /opt/tomcat/latest/webapps.
2. Il sistema eIDAS utilizzerà il servizio tomcat rendendo accessibile le varie componenti ai path http://eidas.org:8080/(SP,IdP,EidasNode,SpecificConnector,SpecificProxyService).
3. Per testare e validare l'installazione: punto 4.1.5.4 del file "eIDAS-Node Installation Quick Start Guide v2.6.pdf" scaricabile dalla stessa pagina di download di eIDAS 2.6.

A.2.6 Modifiche eIDAS

1. Installazione Maven per gestione Maven project.

```
sudo apt install maven
```

2. Per riconfigurare alcune parti di eIDAS (ad esempio modifica ai file che lo compongono) e creare il relativo .war eseguire il seguente comando dalla cartella root EIDAS-Sources-2.6.0.

```
mvn clean install -f EIDAS-Parent/pom.xml -P {NodeOnly, DemoToolsOnly}
```

NodeOnly per ottenere nella cartella EIDAS-Node/target il file EidasNode.war; DemoToolsOnly per ottenere in EIDAS-IdP-1.0/target, EIDAS-SP/target, EIDAS-SpecificConnector/target e EIDAS-SpecificProxyService/target rispettivamente file IdP.war, SP.war, SpecificConnector.war e SpecificProxyService.war.

```
# esempio con skip dei test
mvn clean install -DskipTests -f EIDAS-Parent/pom.xml -P DemoToolsOnly
```

3. Ottenuti quindi i file .war è possibile ripetere la procedura al punto "Deploy su Tomcat" utilizzando però i file .war ottenuti nelle rispettive cartelle e riavviare tomcat per rendere effettive le modifiche.

```
sudo service tomcat restart
```

A.2.7 Extra

1. Per accedere alla pagina di gestione di tomcat è possibile collegarsi a http://localhost:8080/manager con credenziali admin:tomcat.

A.3 Installazione di SPID con OIDC

1. Download repository di Developer Italia (AGID).

   ```
   git clone https://github.com/italia/spid-cie-oidc-django.git
   ```

2. Installazione pip per python (la versione di python cambia in funzione di quella preinstallata sul sistema).

   ```
   sudo python3.7 -m pip install pip
   ```

3. Installazione pacchetti python pip.

   ```
   cd spid-cie-oidc-django
   pip install --upgrade pip
   pip install -e .
   pip install "design-django-theme==v1.4.8"
   ```

4. Configurazione python e avvio server: il sistema offre 3 entità/progetti: relying_party, provider, federation_authority (comprende rp, op e un sistema di management della federazione). In particolare configurare la federation_authority, progetto più completo per lo scopo.

   ```
   cd examples/federation_authority
   cp federation_authority/settingslocal.py.example federation_authority/
       settingslocal.py
   # then customize (optional) federation_authority/settingslocal.py
   ./manage.py migrate
   # load the demo configuration
   ./manage.py loaddata dumps/example.json
   # create a super user
   ./manage.py createsuperuser
   # run the web server
   ./manage.py runserver 0.0.0.0:{8000, 8001, 8002} # porta 8000 per la
       federation_authority
   ```

5. In alternativa il server può essere configurato per l'avvio con docker-compose.

   ```
   sudo apt-get install docker.io
   sudo apt-get install docker-compose
   cd spid-cie-oidc-django
   bash docker-prepare.sh
   sudo docker-compose up
   ```

6. Modifica file hosts per raggiungere SPID con OIDC ai domini trust-anchor.org, relying-party.org o cie-provider.org, in funzione del progetto avviato.

   ```
   vi /etc/hosts
   -> 127.0.0.1 localhost trust-anchor.org relying-party.org cie-provider.
       org
   ```

7. Aprire il browser e puntare a http://trust-anchor.org:8000/oidc/rp/landing per atterrare sulla pagina del Relying Party del progetto federation_authority (trust-anchor.org:8000).

8. Per effettuare un giro di prova è possibile selezionare "Entra con SPID" ed effettuare il login con admin:oidcadmin oppure user:oidcuser.

A.3.1 Extra

La federation authority offre altri strumenti per la gestione tra cui:

- Tool per la decodifica dei JWT raggiungibile al path http://trust-anchor.org:8000/onboarding/tools/decode-jwt;
- Schemas per i vari endpoint raggiungibili al path http://trust-anchor.org:8000/onboarding/schemas/*;
- Dashboard di gestione dell'intera federation authority (gestione di RP, OP e relativi metadata/chiavi private, pubbliche, sessioni OIDC, token emessi, users e relativi attributi, ...) raggiungibile al path http://trust-anchor.org:8000/admin con accesso admin:oidcadmin.

A.4 Personalizzazioni

Il codice eIDAS personalizzato può essere scaricato da https://www.dropbox.com/sh/1d86hdf3da4z54s/AAB26D4Eokf5bh4k6LFn5NxUa?dl=0.

A.4.1 Configurazioni sistema SPID OIDC

```
/home/user/Desktop/spid-cie-oidc-django/examples-docker/federation_authority/
    dumps/example.json
```

Contiene tutte le configurazioni relative a Relaying Party, Provider e Users per SPID OIDC. In particolare aggiunta user gmorabito:oidcuser con i relativi attributi.

A.4.2 Configurazioni eIDAS Specific Proxy Service

```
/opt/tomcat/latest/work/EIDAS-Sources-2.6.0/EIDAS-Config/tomcat/
    specificProxyService/specificProxyService.xml
```

Sostituzione Identity Provider, da quello di demo di eIDAS a SPID con OIDC. Da

```
<entry key="idp.url">http://localhost:8080/IdP/AuthenticateCitizen</entry>
```

a

```
<entry key="idp.url">http://trust-anchor.org:8000/oidc/op/authorization</entry
    >
```

A.4.3 Dipendenze JAVA eIDAS Specific Proxy Service

```
/home/user/Desktop/eIDAS-node-2.6.0/EIDAS-Sources-2.6.0/EIDAS-
    SpecificProxyService/pom.xml
```

Aggiunta dipendenze per il Maven project per l'utlizzo di librerie nei file di codice java. In particolare relative ad algoritmi criptografici e gestione di JSON necessari per i flussi OIDC.

A.4.4 Modifiche al servlet AfterCitizenConsentRequest di eIDAS Specific Proxy Service

`/home/user/Desktop/eIDAS-node-2.6.0/EIDAS-Sources-2.6.0/EIDAS-SpecificProxyService/src/main/java/member_country_specific/specific/proxyservice/servlet/AfterCitizenConsentRequestServlet.java`

Servlet disponibile al path /SpecificProxyService/AfterCitizenConsentRequest.

Gestisce la user consent dello Specific Proxy Service per gli attributi richiesti dal SP - Specific Connector - eIDAS Node - Specific Proxy Service. Se la consent è ok allora prepara una richiesta contente gli attributi richiesti da eIDAS conforme alle esigenze (nel nostro caso una authentication request per OIDC) e la inoltra all'Identity Provider (SPID) authorization endpoint settato nel file di configurazione.

- method: **execute**. Modifica per creare la authentication request in modo conforme a SPID OIDC.

  Partendo dalla specific request creata da eIDAS (contenente gli attributi richiesti dal SP) genera una custom authentication request per SPID contenente come parametri:

  - client_id
  - scope
  - response_type
  - code_challenge a partire da un code_verifier (necessario successivamente durante la richiesta dalla RP al token endpoint dell'IdP per dimostrare di essere l'autrice della richiesta iniziale)
  - code_challenge_method
  - nonce (per evitare attacchi reply all'IdP)
  - state (per tracciabilità del flusso OIDC tra RP e IdP)
  - request (nella forma di un JWS in cui il payload contiene tutti gli attributi richiesti della richiesta eIDAS, ma rappresentati in formato SPID grazie al metodo generatePayload(nonce, state, code_challenge, - richiesta originale eIDAS -) e firmato con il metodo generateJWS(header, payload))

- method: **generatePayload**. Produce un payload conforme alle richieste SPID.
- method: **generateJWS**. Genera il JWS da inoltrare come valore della request all'IdP SPID.
- method: **signJWS**. Produce la firma del JWS, con la chiave privata RSA della RP.
- method: **generateCodeChallenge**. A partire da una stringa random, il code_verifier, genera un code_challenge secondo le specifiche OAuth PKCE.

A.4.5 Modifiche alla JSP (Java Server Page) idpRedirect di eIDAS Specific Proxy Service

`/home/user/Desktop/eIDAS-node-2.6.0/EIDAS-Sources-2.6.0/EIDAS-SpecificProxyService/src/main/resources/META-INF/resources/idpRedirect.jsp`

Pagina JSP indirizzata dal servlet AfterCitizenConsentRequest al termine del metodo execute. Modifica dei campi trasmessi dal form (implicito con submit automatico al caricamento) affichè la richiesta che giunge dall'authorization endpoint dell'IdP SPID contenga effettivamente i parametri sopraindicati.

A.4.6 Creazione nuovo servlet OIDCServlet in eIDAS Specific Proxy Service

`/home/user/Desktop/eIDAS-node-2.6.0/EIDAS-Sources-2.6.0/EIDAS-SpecificProxyService/src/main/java/member_country_specific/specific/proxyservice/servlet/OIDCServlet.java`

Servlet generato per la gestione del flusso OIDC. Disponibile al path /SpecificProxyService/OIDCServlet utilizzato come redirect_uri/callback nella request all'IdP, affinchè, dopo il login dello user, l'IdP OIDC SPID possa inviare l'authorization code direttamente a questo endpoint.

Riceve una GET dall'authorization endpoint dell'IdP: code (authorization_code), state che verifica essere corrispondente a quello inviato nella request e iss che deve essere uguale all'url dell'IdP.

Predispone con il metodo constructRequestBody, una richiesta per il token endpoint dell'IdP, al fine di scambiare l'authorization code ottenuto con un access_token. Effettua quindi la chiamata al token endpoint (trasparente all'utente) ed ottiene un'access_token.

Predisponde quindi una request allo userinfo endpoint dell'IdP (trasparente all'utente) in cui scambierà il token ottenuto, passato come Authorization header Bearer, per gli attributi dell'utente. Ottiene in risposta dallo userinfo endpoint un JWT in forma di JWE. Si occupa quindi di decriptare la chiava simmetrica effimera per perfect-forward-secrecy (in posizione 2 del JWE, dopo l'header che è solo base64 encoded) utilizzando la chiave privata RSA del RP. Impiegando solo gli ultimi 32 byte della chiave esegue infine la decryption con AES (adottando anche l'IV in posizione 3 del JWE) del cyphertext (in posizione 4 del JWE) ed ottiene il JWS con le informazioni dello user.

Con il metodo verifySignature verifica la firma del JWS utilizzando la chiave pubblica dell'IdP e decodifica il payload del JWS contenente gli attributi.

Crea infine una response conforme agli standard eIDAS contenente le info sullo user e la inoltra al servlet IdPResponse che si occuperà poi di rimandarla indietro al nodo eIDAS e quindi allo Specific Connector e al SP originale.

- method: **constructRequestBody**. Crea una request per il token endpoint dell'IdP contenente anche l'authorization code ricevuto ed il code_verifier relativo al code_challenge inviato nella authentication request iniziale affinchè l'IdP posso ulteriormente provare l'identità del RP. Nella request è anche presente una client_assertion in forma di JWS firmata con la chiave privata RSA del RP per autenticità.
- method: **method: createResponse**. Produce una response conforme a eIDAS.
- method: **method: decryptJWE**. Decripta il JWE ricevuto dallo userinfo endpoint ed estrarre il relativo JWS.
- method: **method: decryptKey**. Decripta la chiave effimera trasmessa nel JWE utilizzando la chiava privata del RP.
- method: **method: verifySignature**. Verifica che la signature del JWS estratto dal JWE proveniente dallo userinfo endpoint sia valida, utilizzando la chiave pubblica dell'IdP.

A.4.7 Creazione nuova JSP oidcRedirect in eIDAS Specific Proxy Service

`/home/user/Desktop/eIDAS-node-2.6.0/EIDAS-Sources-2.6.0/EIDAS-SpecificProxyService/src/main/resources/META-INF/resources/oidcRedirect.jsp`

Pagina JSP reindirizzata dall'OIDCServlet per trasmettere la response in formato eIDAS al servlet IdPResponse dello Specific Proxy Service che la inoltrerà poi all'eIDAS node, quindi allo Specific Connector e infine al SP d'origine.

Appendice B

User manual

B.1 Guida per testare l'integrazione

1. Avviare la macchina virtuale linux configurata con eIDAS e SPID.

2. Avviare SPID OIDC con il framewokr fornito da Docker-compose. Dovrebbe impiegare qualche decina di secondi per avviare i progetti e le relative applicazioni esponendo la federation authority sulla porta 8000.

   ```
   cd /home/user/Desktop/spid-cie-oidc-django
   sudo docker-compose up
   ```

3. Il web server Tomcat, per seguendo la configurazione fornita, dovrebbe avviarsi automaticamente al login, tuttavia, potrebbe essere necessario attendere qualche istente affiché sia del tutto pronto. Se dovesse risultare necessario riavviarlo, è possibile eseguire il seguente comando:

   ```
   sudo service tomcat restart
   ```

4. Aprire il browser Firefox all'indirizzo http://eidas.org:8080/SP che rappresenta l'endpoint per atterrare sul demo SP di eIDAS, simulazione di un SP a cui tenta di accedere un utente in un paese europeo diverso da quello di origine.

5. Completare il form della pagina su cui si atterra inserendo il valore CA (Country A) nei primi due box. In realtà, il SP, si dovrebbe trovare in un country A ed il cittadino, dovrebbe avere nazionalità in un contry diverso, come ad esempio country B, ma per motivi di demo, in questo caso, il valore CA, rappresenta un riferimento al web server in uso,determinato dalla relativa porta, quindi Tomcat porta 8080 nello scenario descritto, su cui appoggiarsi per raggiungere lo Specific Connector e lo Specific Proxy Service di eIDAS.

6. Risulta inoltre possibile configurare, dalla stessa pagina di atterraggio del SP, gli attributi che il SP intende richiedere all'IdP. Mandatory devono sempre essere gli attributi *PersonIdentifier*, *DateOfBirth*, *FamilyName*, *LegalPersonIdentifier*, *LegalName*. Se, infatti, lo user che viene successivamente autenticato con l'IdP SPID, non possiede qualcuno di questi, la richiesta eIDAS risulterà sempre in un esito negativo.

7. Dopo aver cliccato sul pulsante *Submit*, si viene reindirizzati sulla parte di Specific Proxy Service (passando per lo Specific Connector e l'eIDAS Node in modo trasparente all'utente), in cui si chiede allo user di autorizzare gli attributi richiesti ed eventualmente, cliccando su *Next*, aggiungerne altri tra quelli configurati precedentemente come Optional.

8. Dopo aver dato il consenso, si attera quindi sulla pagine di SPID per il login (purtroppo il sistema di demo supporta solo l'indicazione di livello spidL2). Effettuare il login con l'utente user:oidcuser o admin:oidcadmin forniti di default, oppure con un'utenza configurata nel

file example.json del sistema SPID. Di default, i due utenti user e admin non possiedono l'attributo mandatorio *dateOfBirth*, quindi senza modifiche aggiuntive, la richiesta terminerà sempre in un errore prima della visualizzazione finale degli attributi al SP.

9. Fornire il consenso per utilizzare gli attributi SPID.
10. Si viene a questo punto nuovamente indirizzati verso lo Specific Proxy SErvice in cui vengono mostrati gli attributi popolati, con ulteriore richiesta di conferma.
11. Infine, si atterrà nuovamente indietro su una pagina di riepilogo del SP, in cui gli attributi popolati vengono visualizzati e sono a questo punto fruibili da parte del SP che li ha chiesti originariamente.

Bibliografia

[1] Paul A. Grassi, Michael E. Garcia, James L. Fenton, "Digital Identity Guidelines", NIST Special Publication, 800-63-3, June 2017, DOI 10.6028/NIST.SP.800-63-3

[2] API academy Certification, "API Security Architect Certification", https://apiacademy.co/2020/07/api-security-architect-certification/

[3] OAuth 2.0, https://oauth.net/2/

[4] OAuth, "ID Tokens", https://www.oauth.com/oauth2-servers/openid-connect/id-tokens/

[5] OpenID Connect, https://openid.net/connect/

[6] Oracle, "Che cosa si intende per SAML (Security Assertion Markup Language)?", https://www.oracle.com/it/security/cloud-security/what-is-saml/

[7] Auth0, "SAML vs OAuth", https://auth0.com/intro-to-iam/saml-vs-oauth/

[8] Commissione Europea, "Identità digitale europea", https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/european-digital-identity_it

[9] Commissione Europea, "eID", https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eID

[10] Mutually Human, "Choosing an SSO Strategy: SAML vs OAuth2", https://www.mutuallyhuman.com/blog/choosing-an-sso-strategy-saml-vs-oauth2/

[11] SPID, "La tua identità digitale", https://www.spid.gov.it

[12] AGID, "Regole tecniche SPID", https://www.agid.gov.it/sites/default/files/repository/_files/circolari/spid-regole/_tecniche/_v1.pdf

[13] AGID, "The eIDAS login", https://www.eid.gov.it/abilita-eidas

[14] AGID, "Nodo eIDAS italiano", https://github.com/AgID/eidas-italian-node

[15] Ed. D. Hardt, "The OAuth 2.0 Authorization Framework", RFC-6749, October 2012, DOI 10.17487/RFC6749

[16] W. Denniss, Google, J. Bradley, Ping Identity, "OAuth 2.0 for Native Apps", RFC-8252, October 2017, DOI 10.17487/RFC8252

[17] M. Jones, Microsoft, D. Hardt, Independent, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC-6750, October 2012, DOI 10.17487/RFC6750

[18] N. Sakimura, Nomura Research Institute, J. Bradley, Ping Identity, N. Agarwal, Google, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC-7636, September 2015, DOI 10.17487/RFC7636

[19] Red Hat, "Cos'è un'API REST?", https://www.redhat.com/it/topics/api/what-is-a-rest-api

[20] Diana Gratiela Berbecaru, Antonio Lioy, Cesare Cameroni, "On Enabling Additional Natural Person and Domain-Specific Attributes in the eIDAS Network", IEEE Access, July 2021, DOI 10.1109/ACCESS.2021.3115853

[21] Developers Italia, "Iniziano i lavori per gli SDK OpenID Connect unificati di SPID e CIE", https://developers.italia.it/it/news/2022/03/15/iniziano-lavori-SDK-SPID-CIE

[22] AGID, "Linee Guida OpenID Connect in SPID", https://docs.italia.it/AgID/documenti-in-consultazione/lg-openidconnect-spid-docs/it/bozza/index.html

[23] Developers Italia, "The SPID/CIE OIDC Federation SDK, written in Python", https://github.com/italia/spid-cie-oidc-django

[24] Commissione Europea, "eIDAS-Node version 2.6", https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eIDAS-Node+version+2.6

[25] Estonian Government "EE specific eIDAS proxy service", https://github.com/e-gov/eIDAS-SpecificProxyService
[26] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC-7515, May 2015, DOI 10.17487/RFC7515
[27] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC-7516, May 2015, DOI 10.17487/RFC7516
[28] Jones, M., "JSON Web Algorithms (JWA)", RFC-7518, May 2015, DOI 10.17487/RFC7518
[29] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC-7519, May 2015, DOI 10.17487/RFC7519
[30] B. Campbell, Ping Identity, C. Mortimore, Salesforce, M. Jones, Microsoft, "Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC-7522, May 2015, DOI 10.17487/RFC7522