**Šta je Single Sign On (SSO) ?**

Single Sign On je metoda koja omogućava korisnicima da se autentikuju preko vise aplikacija ili vebsajta, samo jednom koristeci iste kredencijale (na primer unosom korisnickog imena I lozinke).

Single Sign On autentikacija je u sada postaje tražena više nego ikada. U današnje vreme skoro svaka aplikacija ili vebsajt zahteva neku formu autentikacije da bi imala pristup zaštićenim resursima. Kako broj aplikacija I servisa raste tako potreba za centralizovanim sistemom prijave postaje neophodna.

Koncept centralizovanog sistema prijave je poznat kao Ujedinjeni sistem identiteta (Federated Identity).

Ujedinjeni sistemi identiteta rešavaju nekoliko problema:

* Autentifikaciju
* Pravo pristupa resursima
* Razmena korisnickih podataka
* Upravljanje korisnicima

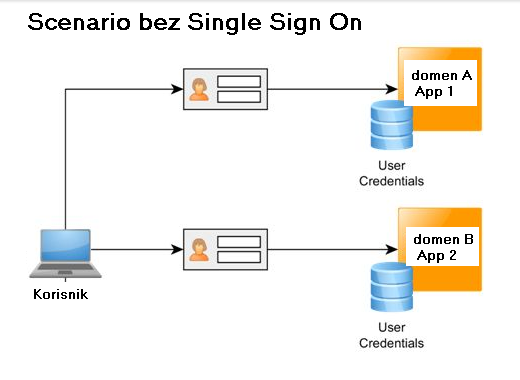
Aspekt Autentifikacije bavi se potvrđivanjem i validiranjem korisničkih podataka (kredencijala) i utvrđivanjem identiteta korisnika.

Pravo pristupa resursima kao sto ime kaze se bavi restrikcijama pristupa odredjene aplikacije odredjenim resursima ( na primer da li aplikacija ili korisnik ima pravo da pristupi nekom resursu)

Razmena korisničkih podataka bavi se razmenom podataka kroz različite sisteme za upravljanje korisnicima.Na primer polja kao sto su “Pravo ime“ mogu biti prisutna u više sistema.Ujedinjeni sistem identiteta sprečava dupliciranje podataka povezivanjem povezanih atributa (linking)

Upravljanje korisnicima je povezano sa samom administracijom (registracija novog korisnika, Brisanje ili Ažuriranje postojeceg korisnika)

Single Sign on je strogo povezan autentifikacionom delu ujedinjenog sistema identiteta.

Kada se korisnik jednom autentikuje unosom kredencijala, preko ujedinjenog sistema identiteta se podaci o samom identitetu dele izmedju podsistema koji zahtevaju podatke.

U razvoju veb aplikacija pre ili kasnije se javlja potreba da korisnik koji se prijavio preko domena A bude automatski prijavljen i na aplikaciji sa domenom B (to je i svrha Single Sign On).

Bez Single Sign On mehanizma korisnik bi morao da se autentikuje unesenjem kredencijala na svakoj aplikaciji (domenu) kojoj pristupa , bez obzira sto te aplikacije cine jednu vecu logicku celinu.

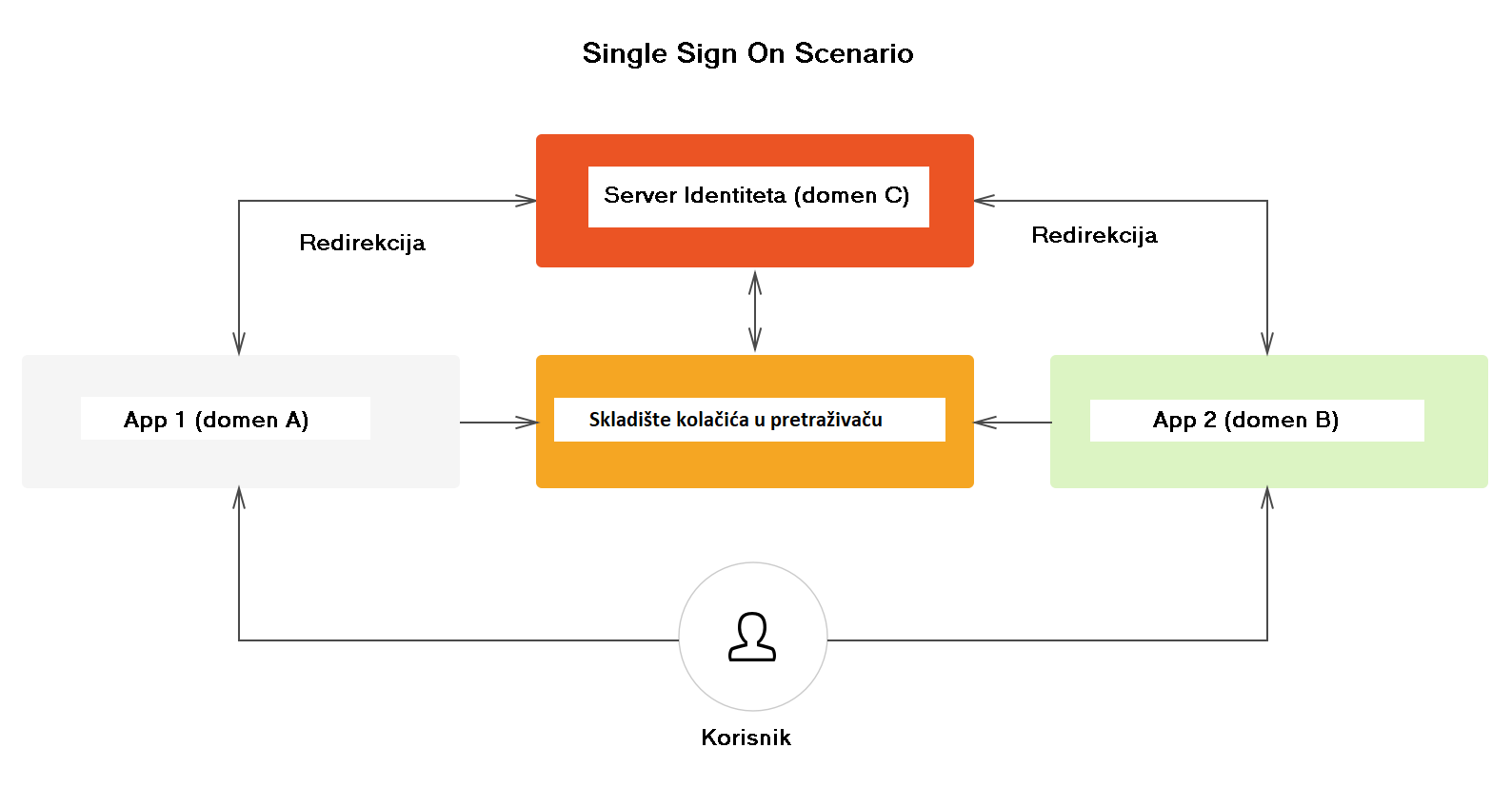
Ocigledno resenje ovakvog problema jeste da informacije sesije(identiteta) dele izmedju različitih domena. Međutim iz sigurnosnih razloga veb pretrazivaci forsiraju politiku zvanu Ista politika porekla.

Ova politika definise da kolačići (ili drugi podaci) mogu da dostupni samo njihovom vlasniku.

Drugim rečima domen A ne moze da pristupi kolačićima domena B i obrnuto.

Postoje različiti Single Sign On protokoli koji omogućavaju deljenje informacija sesije (kolačića) , ali sustinski koncept je isti. Postojanje Ujedinjenog Sistema Identiteta (na posebnom domenu C) preko koga je izvrsena autentikacija i preko koga se sesija deli sa ostalim domenima na neki nacin.

Kada korisnik ode na aplikaciju sa domenom A koja zahteva autentikaciju,automatski biva redirektovan na domen C ( Ujedinjeni sistem identiteta) . Kako je korisnik vec prosao proces autentikacije na domenu C , na primer preko domena B , korisnik ce biti vracen na originalni domen A sa potrebnim informacijama identiteta bez potrebe za ponovnim unosom kredencijala.



Trenutno najpoznatije protokoli za Single Sign On su: OpenID Connect, Facebook Connect, Security Assertion Markup Language (SAML), Microsoft Account.

Za dalji rad izabracemo implementaciju OpenID connect (OIDC) protokola za postizanje Single Sign On.

Nakon uspesne autentikacije za dozvolu prava pristupa odredjenim resursima koristiće se implementacija Oauth2 protokola (OAuth 2.0 is the industry-standard protocol for authorization)

**Prednosti Single Sign On**

Korisnicima je glavna prednost što više ne moraju da pamte na destetine lozinki.Ovo čak oslobađa korisnika od upravljanja pojedinacnih lozinki na svakoj aplikaciji ponaosob, zbog čega se SSO prijave takođe smatraju i menadzerima lozinki.Smatrajući da je ovo tako zgodna alternativa i zato sto štedi vreme i gnjavažu, SSO rešenja se koriste kako u privatnom tako i u profesionalnom kontekstu.

Kompanije koje primenjuju SSO u svom radu očekuju veću produktivnost svijih zaposlenih i manje poziva službe za pomoc zbog zaboravljenih lozinki.Zbog toga IT ima manje posla i manje košta.Istovremeno IT osoblju olakšava dodeljivanje novih naloga novim zaposlenima ili brisanje pristupa bivšim zaposlenima.

Takođe lozinku treba uneti samo na jednom interfejsu, a to smanjuje potencijal za napade (pecanje) u sistemu. Pod ovim okolnostima kompanija moze sebi priuštiti da usresredi rad na zastiti samo na jednom mestu.Kao sto su na primer SSL sertifikati.

Povećava brzinu tamo gde je napotrebnija. U okruženjima kao sto su bolnice,odbrambena industrija I hitne službe, gde veliki broj ljudi I odeljenja zahteva brz I nesmetan pristup istim aplikacijama.

**Problemi Single Sign On**

Sa druge strane postoje određeni problemi i napori u primeni, kao i slabosti samog SSO.

U osnovi mogu se koristiti samo one usluge koje podržava odgovarajući sistem SSO.Ako sistem zakaže pristup povezanim aplikacijama biće ograničen ili nemoguć. U ovom slučaju SSO postaje takozvano usko grlo.

Takođe treba uzeti u obzir i sigurnost SSO prijave.Ako korisnik napusti svoju aplikaciju a da nije izvršio odjavljivanje, treća strana bi teoretski mogla da iskoristi vreme do automatske odjave (Single Sign Out), kako bi iskoristila pristup odobren putem SSO prijave.Takođe je problematicno ako “glavna lozinka“ za SSO interfejs padne u pogrešne ruke.To napadaču daje trenutni pristup svim povezanim uslugama.

Postoji zabrinutost i zbog GDPR (General Data Protection Regulation) a, u kojem su zahtevi za zaštitu ličnih podataka pooštreni u celoj Evropi od 25. Maja 2018. godine. Sada je potrebno dobiti izričit odgovor od korisnika da bi mogli da koriste SSO prijavu.U prošlosti je bio potreban i ovaj sporazum, ali su zakonitosti toliko promenjene da je situacija sada mnogo stroža.

S obzirom na ove potencijalne rizike, potrebno je obratiti posebnu pažnju na sigurnost podataka koji se čuvaju na strani servera.Ima smisla povećati sigurnost SSO funkcija koristeci na primer dvofaktorsku potvrdu identiteta (two-factor authentication) ili druga rešenja kao sto su pametne kartice (smart cards) ili tokeni.

Ako neko istraži SSO prijavu na Internetu, pronalazi relativno malo negativnih informacija o postupku višestruke potvrde identiteta.Umesto toga, godinama se to tretira kao sasvim prikladno otkriće za digitalno radno mesto, što se tiče udobnosti i sigurnosti podataka.

US-American Cloud Access Security Broker (CASB) hvali globalnu upotrebu usluga u oblaku u kompanijama, ali istovremeno kritikuje relativno nisku upotrebu metoda za za višestruku potvrdu identiteta (SSO prijava)

**OIDC i OAuth2**

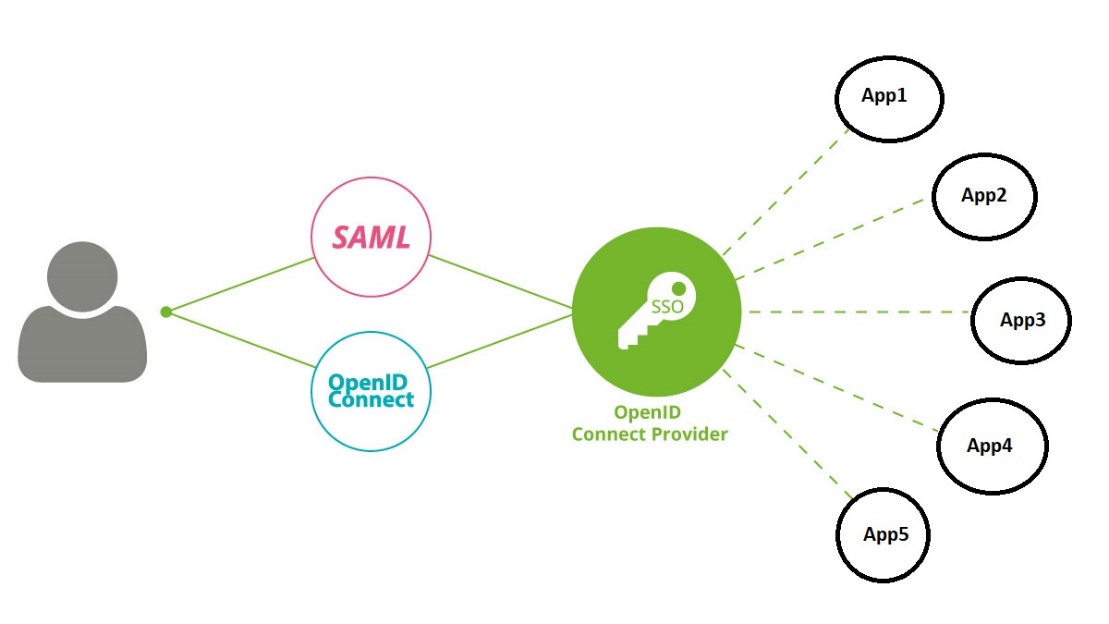
U predhodnom odeljku smo objasnili koncept Single Sign On (SSO) -a i rekli smo da postoje različiti protokoli za implementaciju SSO-a (OpenIdConnect, Security Assertion Markup Language).Kao i da postoji potreba za centralizovanim sistemom prijave, odnosno Serverom Identiteta (IdP- Identity Service Provider).

Takođe smo rekli da ćemo za dalji rad izabrati implementaciju OIDC i OAuth2 protokola.Pre nego sto uđemo u detalje kako rade ovi protokoli, ukratko cemo objasniti razlike izmedju njih.

Suštinska razlika izmedju ovih protokola je ta sto se OAuth2 koristi za kontrolu prava pristupa zaštićenim resursima (neka aplikacija ili web Api ili datoteka), dok su OIDC i SAML industrijski standardi za potvrdu identeta odnosno autentifikaciju. OAuth2 se koristi u potpuno drugačijim situacijama od druga dva standarda, a može se koristiti istovremeno u kombinaciji sa OIDC-om i SAML-om, što uglavnom i jeste slučaj.

OAuth2 omogućava aplikacijama pristup zaštićenim resursima korisćenjem tokena zvanim Access Tokens.

OIDC je izgradjen na vrhu OAuth2 protokola i koristi dodatni token u JWT formatu,koji se naziva ID Token.Posebno je fokusiran na autentikaciju korisnika i široko se koristi za omogućavanje korisničkih prijava na veb sajtovima i u mobilnim aplikacijama.S obzirom da je OIDC nadogradnja OAuth2 protokola, predstavlja potpuno razvijen protokol za autentikaciju i dodelu prava pristupa (autorizaciju), koji intenzivno koristi sigurnosne JSON tokene za komunikaciju izmedju servisa, aplikacija i servera identiteta (IdP).

SAML je nezavisan od OAuth2 protokola, i koristi XML format za razmenu poruka za potvrdu identiteta,za razliku od JWT formata koji koristi OIDC.Češće se koriti za pomoć korisnicima korporacija da se prijave u više aplikacija pomoću jedne prijave. 

**Upoređenje ključnih razlika između OIDC i SAML protokola:**

Oba protokola se koriste za postizanje istih ciljeva, međutim sam nacin kako autentikuju korisnika se razlikuje, korišcenjem drugačijih tehnologija i metoda.

**RP** **nasuprot SP :** Oba protokola, RP ( Relying Party, termin za klijentsku aplikaciju) u terminologiji OIDC-a, i SP (Service Provider) u slucaju SAML, redirektuju korisnika serveru identiteta (IdP ili OP).Razlika je u tome sto SAMP nije bas kompatibilan sa određenim tipom aplikacija, kao sto su na primer mobilne aplikacije, u poređenju sa OIDC-om koji jednako dobro funkcionise kako sa veb tako i sa mobilnim aplikacijama.

**Format Poruka :** SAML koristi tvrdnje koje predstavljaju attribute autentikacije kao I takozvane izjave za dodelu prava pristupa I formatirane su korišćenjem XML formata. Sa druge strane OIDC koristi JSON Web tokene pod nazivom ID Tokeni u JWT format, koji pruža informacije za potvrdu identiteta.

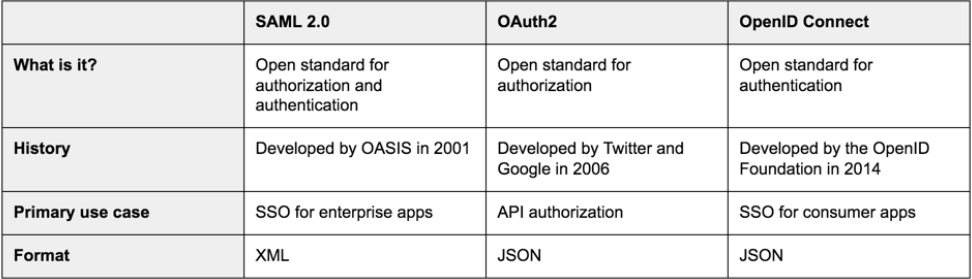
**Podrška za API :**  OIDC je nastao iz razloga što je SAML Xml format “Težak”, I ne moze se dobro integrisati sa Api-ima. OIDC radi tako što koristi HTTP komunikacioni kanal za slanje “Lakih” JSON sigurnosnih tokena za potvrdu identiteta korisnika. Sa druge strane SAML koristi SOAP, koji je takođe protokolski sloj preko HTTP-a, međutim šalje “Teške” poruke u XML format za potvrdu identiteta korisnika.

**Autentikacija korišćenjem mobilnih aplikacija :** OIDC koristi RESTFUL komunikaciju za stvaranje i prosleđivanje “Laganih“ JSON tokena, koji se prosleđuju između servera identiteta (IdP) i pouzdane strane (relayng party) odnosno klijentske aplikacije, što čini OIDC vodećim protokolom za mobilne aplikacije. Za razliku od SAML-a koji je uglavnom razvijen za autentikaciju veb aplikacija, a budući da koristi XML, nema prostora za buduću upotrbu u mobilnim aplikacijama, što daje prednost OIDC-u.

**Korisnikov Pristanak :** S obzirom da je OIDC izgrađen na vrhu OAuth2 protokola za podršku autentikacije, može pružiti ugrađeni sloj za prava pristupa (Autorizacije) , koji podstiče korisnika da prvo pristane na ono što pružalac usluga (IdP) može obezbediti odnosno dati pristup.SAMl takođe može obezbediti tok saglasnosti, međutim to čini kodiranjem samog programera (“Hard-Coding“), umesto da obezbedi tok saglasnosti kao deo standarda u svom protokolu.

**Statična Autentikacija :**  U SAML protokolu Server Identiteta (IdP) i Pouzdana strana (relayng party) moraju biti konfigurisani da se poznaju pre stvarnog prenosa podataka. Sa druge strane za OIDC to nije slučaj.

**Implementacija :** OIDC protokol je daleko jednostavnije implementirati od strane programera , što povecava upotrebljivost samog protokola za različite svrhe. Takođe može biti lako implementiran od nule ili korišćenjem već gotovih biblioteka koje su dostupne kao pomoć i podrška developerima.



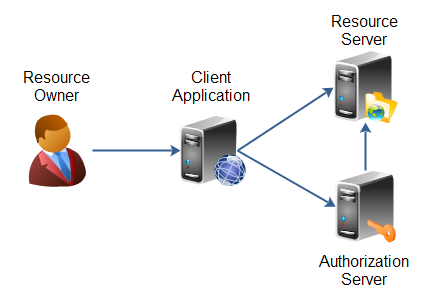
**OpenId Connect (OIDC)**

Rekli smo da je OIDC sloj za potvrdu identiteta (autentikaciju) koji je izgrađen na vrhu (ekstenzija) protokola za autorizaciju (OAuth2). To znači da OIDC koristi iste komponente i arhitekturu kao i OAuth2 ali za potvrdu identiteta (za razliku od OAuth2 koji izdaje access tokene za dodelu prava pristupa , OIDC se koristi za izdavanje ID tokena i koristi iste tokove odobrenja (Authorization Grant flows) kao i OAuth2.

OAuth je prvobitno bio zamisljen kao rešenje necega što je poznato kao delegirani pristup “Delegated access“ što predstavlja omogućavanje aplikacijama da međusobno komuniciraju i dele informacije u ime samog korisnika.

Ovaj problem je zahtevao rešenje koje bi standardizovalo nacin na koji dve ili više nepovezanje aplikacije dele lične korisničke podatke. Iako OAuth2 to rešava, on zapravo ne prenosi podatke koji mogu identifikovati korisnika koji trazi povlastice odnosno pravo pristupa (access token ne sadži informacije o samom korisniku vec o tome koja aplikacija ima prava pristupa određenim resursima).OIDC dodaje funkciju potvrde autentičnosti u OAuth -ov program za autorizaciju.

Pre nego što uđemo u detalje tokova odobrenja koji su definisani u specifikaciji OAuth2 protokola, prvo cemo proći i objasniti koncepte i terminologiju koja se koristi u OAuth2 protokolu.



**Koncepti u OAuth 2.0 protokolu**

Sledeći koncepti se generalno koriste u OAuth 2.0:

**Vlasnik Resursa (Resource Owner)**

Entitet sposoban da odobri pristup zaštićenom resursu.Kada je entitet osoba, ona se naziva korisnik.

**OAuth Klijent**

Nezavisna aplikacija koja zeli pristup privatnim resursima vlasnika resursa. OAuth klijent može da salje zahteve resursima u ime vlasnika resursa, nakon sto mu vlasnik dodeli ovlašćenje.OAuth uvodi dve vrste klijenata:

* Poverljivi: Registrovani su kao tajna klijenta (client secret)
* Javni

U terminologiji OIDC-a cesto se za klijente koristi i termin poverljiva strana odnosno aplikacija (Relayng Party RP)

**OAuth server (IdP)**

Poznat kao server za autorizaciju ( U OIDC je poznat po terminologiji Server Identiteta ili OpenID OP).

Server koji OAuth klijentima daje opseg i pravo pristupa zaštićenom resursu u ime vlasnika resursa.Server izdaje token pristupa (access token), klijentu nakon sto uspešno izvrši sledece korake:

* Autentifikuje vlasnika resursa
* Validira zahtev i pravo za odobrenje
* Pribavlja Ovlašćenje vlasnika resursa

Server za autorizaciju takodje može biti i server resursa.

**Token prava pristupa (Access Token)**

String (niz) koji predstavlja ovlašćenje koje je OAuth klijentu dodelio vlasnik resursa. Ovaj niz predstavlja odredjeni opseg kao i trajanje prava pristupa. Odobrava ga vlasnik resursa, a nameće OAuth Server.

**Zaštićeni resurs (Protected Resource)**

Zaštićeni resurs je resurs kome klijent pristupa kada server resursa validira access token , koji klijent salje u zahtevu za odredjenim resursom.

**Server Resursa (Resource Server)**

Server koji hostuje zaštićene resurse.Koristi tokene prava pristupa (access tokens) za validaciju i vracanju resursa klijentu , ukoliko klijent ima prava pristupa na odredjeni resurs.

Server Resursa takodje može biti i OAuth Server.

**Token Osveženja (Refresh Token)**

String (Niz) koji se koristi za dobijanje novog Tokena prava pristupa (access tokena).Token osveženja opciono izdaje server autorizacije, klijentu zajedno sa tokenom prava pristupa. Kada istekne token prava pristupa, klijent može koristiti token osveženja da zatraži novi token pristupa koji se zasniva na istoj autorizaciji bez ponovnog ukljucivanja vlasnika resursa.

**Token Identiteta (ID Token)**

OIDC proširuje mogućnosti autentifikacije OAuth2 tako što ukljucuje takozvani Token identiteta izdat kao Json Veb Token (JWT). ID tokeni su konceptualno analogni ID karticama, jer sadrže skup tvrdnja (claims) o korisniku, poput imena, e-pošte i ostalih licnih podataka korisnika.Ovo se razlikuje od tokena prava pristupa (access token) u tome sto token prava pristupa ne uključuje nikakve informacije identiteta, već postoje za autorizaciju pristupa serverima resursa sa ograničenim opsegom pristupa.Token prava pristupa nije dokaz za potvrdu identiteta, zato što se tokeni prava pristupa mogu dobiti na više nacina.Zato se ID Tokeni koriste za potvrdu Identiteta korisnika.

**Kod Autorizacije (Authorization Code)**

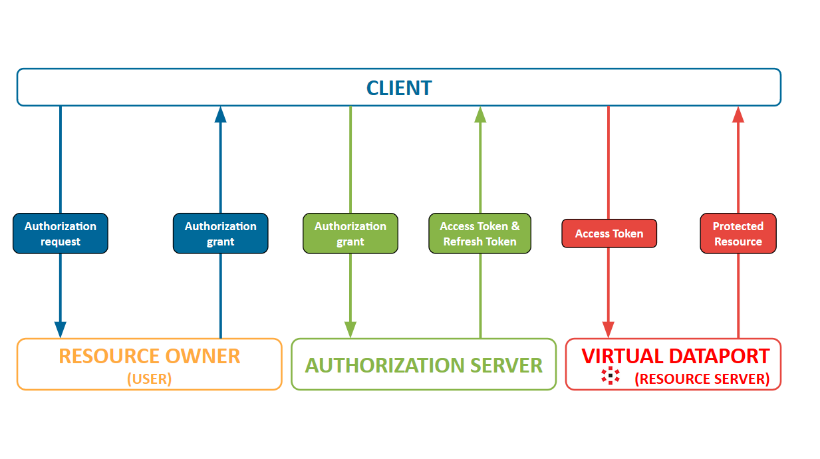
Kod koji server za autorizaciju generiše kada vlasnik resursa odobri zahtev i koristi se za validaciju klijenata pre izdavanja tokena prava pristupa (access token-a)

**Dodela Odobrenja (Authorization Grant)**

Dozvola koja predstavlja ovlašćenje vlasnika resursa za pristup zaštićenim resursima. OAuth2 Klijenti koriste odobrenje za dobijanje tokena (access token, id token, refresh token).Postoje četri tipa odobrenja autorizacije odnosno tokova odobrenja:

* Tok Autorizacionim kodom (Authorization Code Flow)
* Implicitni Tok (Implicit Flow)
* Kredencijali Vlasnika resursa (Resource Owner Password Credentials)
* Kredencijali Klijenta (Client Credentials)
* Hibridni Tok (Hybrid Flow) – Dodatno uveden od strane OIDC sa posebnim namenama
* Tok Autorizacionim kodom sa dokazom kljuca za razmenu kodova (Authorization Code Flow with Proof Key for Code Excange - PKCE)

Važno je napomenuti da ove tokove koristi i OIDC samo dodaje izdavanje tokena identiteta.U nastavku cemo objasniti svaki od toka i koje su njihove prednosti i mane kao i preporučeni tipovi klijentske aplikacije za svaki od navedenih tokova.



S obzirom da je ovde fokus na Single Sign On-u koristeći OIDC protokol, detaljnije cemo objasniti samo tokove koji je izvršavaju uz pomoć pretraživaca i samog korisnika sa ciljem autentikacije i izdavanja tokena identiteta.

U tokovima dozvole autorizacije sa takvom namenom spadaju:

* Tok Autorizacionim kodom (Authorization Code Flow)
* Implicitni Tok (Implicit Flow)
* Hibridni Tok (Hybrid Flow) – Dodatno uveden od strane OIDC sa posebnim namenama
* Tok Autorizacionim kodom sa dokazom kljuca za razmenu kodova (Authorization Code Flow with Proof Key for Code Excange - PKCE)

Objasnićemo kada bi tačno trebalo da se koristi svaki od njih, uključujući neke primere gde bi mogao da se koristi neki tok ali možda izazove više problema nego što ih reši.

Još jedna stvar vredna napomene, pre nego što zađemo u detalje, je da većina pružača usluga identiteta (OIDC OpenId providers) sada nude biblioteke koji omogućavaju upotrebu ove funkcije a da programeri nisu svesni svih detalja niskog nivoa implementacije.

Sada ćemo ukratko napomenuti koji od tokova dozvole su preporučeni za korišćenje od razlicitih vrsta klijentskih aplikacija.

**Veb Aplikacija sa serverskom stranom (Web Application with dedicated server-side component)**

Veb aplikacije sa severskom stranom poput .net MVC-a mogu sigurno cuvati tajnu odnosno client secret (Confidential Client) tako da se za ovu crstu aplikacija preporucuje korišćenje Toka Autorizacionim kodom.

**Desktop Aplikacija (Desktop Native Application)**

Desktop aplikacija obično nema mogućnost da cuva tajnu klijenta tako da se za ovakvu vrstu aplikacije preporucuje Tok Autorizacionim kodom sa dokazom kljuca za razmenu kodova (Authorization Code Flow with Proof Key for Code Excange - PKCE) sa javnim klijentom.

**Mobilna aplikacija (Mobile Native Application)**

Za mobilnu aplikaciju takođe važi da ne moze sigurno da čuva tajnu klienta tako da se i za ovu vrstu aplikacije preporučuje Tok Autorizacionim kodom sa dokazom kljuca za razmenu kodova (Authorization Code Flow with Proof Key for Code Excange - PKCE) sa javnim klijentom.

**SPA i Javascript aplikacije**

Ovakav tip aplikacija se izvršava u korisničkom pretraživaču što takodje ne predstavlja siguran način za čuvanje klijentske tajne. Za ovaj tip aplikacije može se koristiti Implicitni tok , medjutim on ima svoje sigurnosne rizike tako da se preporučuje korišćenje Toka Autorizacionim kodom sa dokazom kljuca za razmenu kodova (Authorization Code Flow with Proof Key for Code Excange - PKCE) sa javnim klijentom

Mnoštvo aplikacija se neće tačno uklopiti u jednu od ovih kategorija. Pojedine situacije moraće se procenjivati na osnovu sopstvenih potreba i zahteva.

**Stvari koje treba uzeti u obzir:**

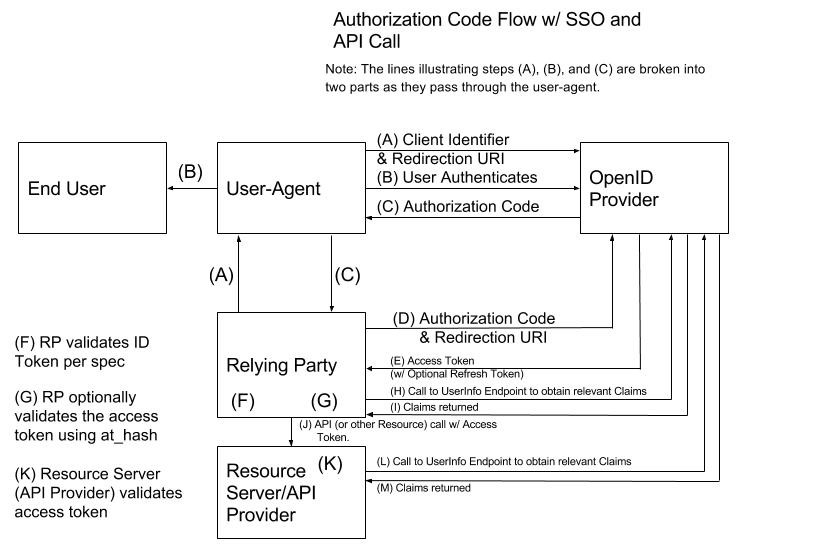
* Javni naspram poverljivih klijenata ( Vi ste vlasnik klijentske aplikacije / kontrolisete aplikaciju u odnosu na nezavisnu aplikaciju odnosno aplikaciju treće strane)
* Tip Klijenata: veb aplikacija, desktop, mobilna, spa aplikacija.
* Identitet korisnika ili aplikacije (ili oboje)
* Želja za centralizovanim upravljackim tokom prijavljivanja

**Tok Autorizacionim kodom (Authorization Code Flow)**

Sada ćemo objasniti kako funkcionise tok autorizacionim kodom i kako se korišćenjem ovog toka korisnik autentikuje preko klijentske aplikacije (Relayng Party) i dobija token identiteta i prava pristupa preko servera identiteta (IdP).

Korišćenjem ovog toka se trebaju obezbediti sledece stvari:

* Autentikacija krajnjeg korisnika
* Autentikacija klijentske aplikacije (Relayng Party RP u OIDC terminologiji), opciono
* Klijentska aplikacija ne vidi kredencijale (šifru) krajnjeg korisnika
* Pretraživač ne vidi tokene koji se prenose izmedju Servera identiteta (IdP) i Klijentske apikacije (Relayng Party)



Na početku korisnik inicira proces prijave tako što u pretraživaču kuca adresu svoje aplikacije (Relayng Party).

Pretraživač tada upućuje zahtev klijentskoj aplikaciji.

**Zahtev za autentikaciju**

Kada klijentska aplikacija (RP) primi zahtev , proverava da li je korisnik prosao proces autentikacije (da li ima izdat token identiteta).Ukoliko nije tada Klijentska aplikacija redirektuje korisnika na Server Identiteta (OpenID Provider OP) koji korisniku vraca stranu za prijavljivanje, gde korisnik treba uneti svoje kredencijale (korisničko ime i lozinku)

GET /oidc/authorize?  
response\_type=code  
&scope=openid%20profile%20email  
&client\_id=s6BhdRkqt3  
&state=af0ifjsldkj  
&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1  
&nonce=fjsjsellesldska...  
Host: server.example.com

Klijentska aplikacija salje zahtev za autorizaciju Serveru Identiteta tako što gadja autorizacionu metodu sa sledećim parametrima:

* **Response\_type=code** Ovim parametrom se specificira da se radi o toku autorizacionim kodom i nakon sto korisnik zavši proces autentikacije , server Identiteta vraca klijentskoj aplikaciji kod sa kojim se kasnije u kombinaciji sa klijentskom identifikacijom i šifrom dobijaju potrebni tokeni.
* **Scope= openid profile email** Scope parametrom se zapravo specificiraju specificna prava odnosno opseg pristupa sa kojima se izdaju tokeni i u sklopu OIDC-a mozemo zahtevati samo specificne informacije koje ce biti izdate u tokenima. Svaki scope se navodi sa razmakom u zahtevu, i na primer openid nam kaze da zelimo izdavanje identity tokena samo sa korisnikovim identifikacionim kodom. Profile,email scopom zelimo da u tokenu identiteta dobijemo vise informacija o korisniku poput emaila,slike,imena itd.Važno je napomenuti da se u Serveru identiteta mogu dodavati specificni ospezi i prava za same korisničke informacije inutar id tokena kao i za prava pristupa specifičnim resursima unutar access tokena.
* **Client\_Id** Identifikator klijenta (privatni ili javni) koji je registrovan u serveru identiteta i preko koje zapravo Klijentska aplikacija vrši autentikaciju i autorizaciju.Prilikom registracije klijenata u serveru identiteta se zapravo konfigurišu svi ovi parametri i opsezi pristupa i klijentska aplikacija mora da pruži iste parametre prilikom autorizacije.
* **State** Ovaj parametar uglavnom sluzi da sprči takozvane CSRF (Cross-site request forgery) napade I da se dodatno validira zahtev ka serveru identiteta. Klijentska aplikacija šalje random state parametar , nakon autentikacije server identiteta, klijentskoj aplikaciji vraća isti state kako bi utvrdila da je zapravo to odgovor na njen zahtev.
* **Redirect\_uri** Url adresa klijentske aplikacije na kojoj server identiteta salje kod nakon autentikacije.
* **Nonce** Dodatna zaštita od napada.Klijent šalje nonce parametar, koji server identiteta ubacuje kao tvrdnju unutar tokena identiteta. Klijent kada primi token proverava da li se nonce vrednost slaže sa poslatom.Uvedeno je radi sprečavanja takozvanih replay napada.

Ukoliko autorizaciona metoda unutar servera identiteta utvrdi da korisnik nije prošao potrebnu autentikaciju, odnosno nije uneo korisničko ime ili lozinku (proverava se uglavnom informacija o sesiji odnosno kolačić identiteta da li je setovan) , tada se prvo korisnik redirektuje na stranu za prijavljivanje.

**Uspešan odgovor nakon potvrde identiteta**

Nakon uspešne autentikacije korisnika na strani prijave, Server Identiteta (OP) ce redirektovati korisnika nazad na autorizacionu metodu , sa informacijama o sesiji (uglavnom u formi kolačića). Tada će metoda za autorizaciju vratiti HTTP redirekt (Sa Kodom Autorizacije), klijentskoj aplikaciji.

HTTP/1.1 302 Found  
Location: <https://client.example.org/cb?>  
code=SplxlOBeZQQYbYS6WxSbIA  
&state=af0ifjsld

Kao rezultat , pretraživač šalje GET zahtev klijentskoj aplikaciji u sledećoj formi:

GET <https://client.example.org/cb?>  
code=SplxlOBeZQQYbYS6WxSbIA  
&state=af0ifjsldkj

**Zahtev za dobijanje tokena**

U ovom trenutku, Klijentska aplikacija (RP) je primila kod autorizacije bez da je videla zapravo korisnikove kredencijale (korisničko ime i lozinku). Sada RP treba da dobavi tokene (id token, access token, refresh token) i ostale informacije iz Token metode unutar servera identiteta.

Tako da sada RP salje zahtev Token metodi za pribavljanje potrebnih tokena.

POST /oidc/token HTTP/1.1  
Host: server.example.com  
Content-Type: application/x-www-form-urlencoded  
grant\_type=authorization\_code&  
code=SplxlOBeZQQYbYS6WxSbIA&  
redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb&  
client\_id=s6BhdRkqt3&  
client\_secret=blah\_blah\_blah\_1234

Client\_secret parametar je potreban ukoliko je RP zapravo registrovan kao privatni (confidential) klijent.

Kao što vidimo sada Klijentska aplikacija korišćenjem koda I klijenta zahteva izdavanje potrebnih tokena.

Zbog neophodne validacije osim koda, serveru identiteta se šalju I ostale klijentske informacije koje su registrovane kao deo klijenta (privatnog ili javnog) unutar servera identiteta (grant\_type, redirect\_uri, client\_id, client\_secret).

**Odgovor Token Metode**

Nakon validacije koda i klijentskih parametra, server identiteta (OP) vraća klijentskoj aplikaciji (RP) ptrebne tokene u sledećem obliku:

HTTP/1.1 200 OK  
Content-Type: application/json  
Cache-Control: no-store  
Pragma: no-cache  
{  
“access\_token”: “SlAV32hkKG”,  
“token\_type”: “Bearer”,  
“refresh\_token”: “8xLOxBtZp8”,  
“expires\_in”: 3600,  
“id\_token”: “eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc  
yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5  
NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ  
fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz  
AKfQ.ggW8hZ1EuVLuxNuuIJKX\_V8a\_OMXzR0EHR9R6jgdqrOOF4daGU96Sr\_P6q  
Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ  
NqeGpe-gccMg4vfKjkM8FcGvnzZUN4\_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd  
QyHE5lcMiKPXfEIQILVq0pc\_E2DzL7emopWoaoZTF\_m0\_N0YzFC6g6EJbOEoRoS  
K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3\_osjzw2PAithfubEEBLuVVk4  
XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg”  
}

**Validacija Tokena**

Kada klijentska aplikacija (RP) primi token identiteta (id\_token) on mora biti potvrđen odnosno dodatno validiran prema odeljku 3.1.3.7 OIDC specifikacije (korak F). Iako je opciono savetuje se da se ovaj deo ne preskače zato što ugrožava bezbednost.

U OIDC specifikaciji Id\_token se koristi radi provere identiteta na strani RP-a , dok se access\_token koristi za opcionalno preuzimanje dodatnih informacija o korisniku korišćenjem UserInfo metode na strani servera identiteta (OP) , kao I za upućivanje poziva serverima resursa.

UserInfo metoda se koristi za dobavljanje dodatnih informacija o korisniku ,koja nisu deo tokena identiteta.

GET /connect/userinfo

Authorization: Bearer <access\_token>

HTTP/1.1 200 OK

Content-Type: application/json

{

"sub": "248289761001",

"name": "Bob Smith",

"family\_name": "Smith",

"role": [

"user"

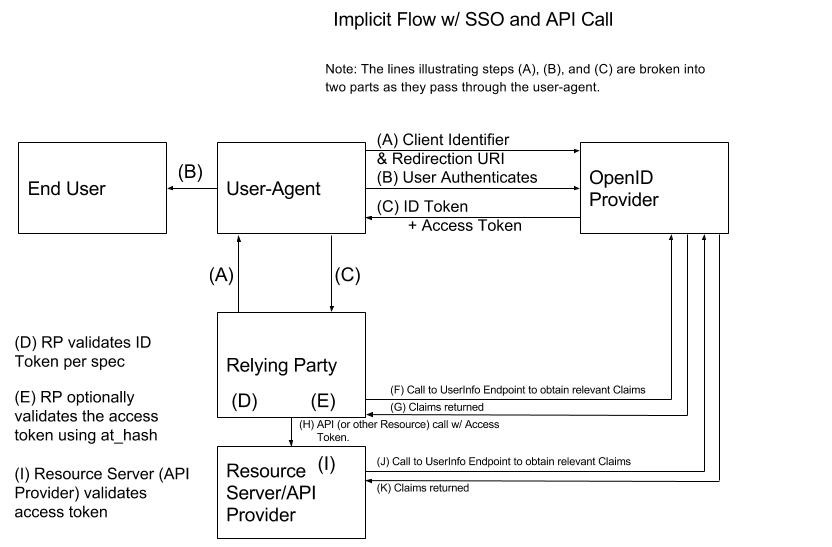
]

}

**Implicitni Tok (Implicit Flow)**

Implicitni tok je slican toku autorizacionim kodom samo je pojednostavljen i prvobitno se preporucivao native i javaskript aplikacijama, gde je token vracen aplikaciji odmah, bez dodatnog koraka za razmenu autorizacionog koda.

S obzirom da native i javascript aplikacije ne mogu sigurno čuvati klijentsku tajnu (client secret), ovaj tok uglavno koristi javne klijente sa strane servera identiteta.



S obzirom da su korici jako slicni toku autorizacionom kodu, fokusiracemo se da obajsnimo samo ključne razlike.

**Zahtev za autentikaciju**

GET /oidc/authorize?  
response\_type=id\_token%20token  
&client\_id=s6BhdRkqt3  
&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb  
&scope=openid%20profile  
&state=af0ifjsldkj  
&nonce=n-0S6\_WzA2Mj HTTP/1.1   
Host: server.example.com

Klijentska strana (Relayng Party) upucuje get zahtev preko pretrazivaca, gađajuci /authorize metodu na strani servera identiteta (OP).

Razlika u ovom zahtevu u odnosu na tok autorizacionim kodom jeste u parametru response\_type gde se ovoga puta navodi id\_token i opcionalno token. Ako se kao parametar unese id\_token to govori serveru identiteta da klijentu vrati token identiteta kao odgovor direktno kroz pretrazivač i da se radi o implicitnom toku. Dodatni opcioni parametar token govori serveru identiteta da klijentska aplikacija takodje zeli i token za prava pristupa (access\_token), koji ce takodje biti vraćen kroz pretrazivač kao parametar.

**Uspešan odgovor nakon potvrde identiteta**

HTTP/1.1 302 Found  
Location: [https://client.example.org/cb#](https://client.example.org/cb)  
access\_token=SlAV32hkKG  
&token\_type=bearer  
&id\_token=eyJ0 ... NiJ9.eyJ1c ... I6IjIifX0.DeWt4Qu ... ZXso  
&expires\_in=3600  
&state=af0ifjsldkj

Server Identiteta odgovara na zahtev u ovom formatu. Kao što mozemo videti ovoga puta se ne vraća kod vec se direktno prosleđuju tokeni.

Zatim Pretraživač šalje redirektovani zahtev, klijentskoj aplikaciji (RP). Zahtev ce izgledati kao na slici ispod.

POST <https://client.example.org/cb>

access\_token=SlAV32hkKG  
 &token\_type=bearer  
 &id\_token=eyJ0 ... NiJ9.eyJ1c ... I6IjIifX0.DeWt4Qu ... ZXso  
 &expires\_in=3600  
 &state=af0ifjsldkj

Klijentska aplikacija (RP) presreće ovaj zahtev i vadi tokene ( i druge opcione parametre)

Važno je napomenuti da se ne preporučuje upotreba implicitnog toka (a neki serveri taj tok u potpunosti zabranjuju) zbog sigurnosnih rizika vraćanja tokena u HTTP preusmeravanju bez ikakve potvrde da ga je klijentska aplikacija primila.

Javni klijenti poput native i javascript aplikacija, sada bi trebalo da koriste Tok Autorizacionim kodom sa dokazom kljuca za razmenu kodova (Authorization Code Flow with Proof Key for Code Excange - PKCE), koji je napravljen da prevaziđe ove sigurnosne probleme za javne klijente servera identiteta.

PKCE ćemo objasniti kasnije.

**Hibridni Tok**

Hibridni tok je kombinacija implicitnog toka i toka autorizacionim kodom. Svi koraci su isti s tim sto se token identiteta vraća implicitnim tokom kroz pretraživač a token prava pristupa preko autorizacionog koda.

Aplikacije koje su u stanju da bezbedno čuvaju klijentske tajne (MVC aplikacije), mogu imati koristi od upotreba hibridnog toka, koji omogućava klijentskoj aplikaciji (RP) trenutni pristup tokenu identiteta, a istovremeno obezbeđuje sigurno preuzimanje tokena za prava pristupa (access\_token, refresh\_token, takodje i novi id\_token) razmenom koda i klijentske tajne.

Ovo može biti korisno u situacijama kada klijentska aplikacija mora odmah pristupiti informacijama o korisniku, ali mora izvršiti određenu obradu i validaciju sa serverom identiteta pre nego što pristupi zaštićenim resursima.

S obzirom da smo objasnili korake u implicitnom toku i toku autorizacionim kodom , sada ćemo samo ukratko proći korake.

* Klijentska aplikacija salje zahtev za autentikaciju i autorizaciju redirekcijom na /authorize metodu serevera identiteta. Ovoga puta parametar response\_type se postavlja na

id\_token code što govori da se radi o hibridnom toku.

* Nakon autentikacije server identiteta vraća klijentskoj aplikaciji kod kao i token identiteta
* Klijentska aplikacija odmah moze da iskoristi token identiteta da bi dobila podatke o klijentu
* Klijentska aplikacija zahteva token za pravo pristupa resursima (access\_token) tako što serveru identiteta šalje kod i klijentsku tajnu radi dodatne validacije. (Gađa /token metodu na strani servera identiteta.
* Nakon što server identiteta validira zahtev preko predhodno poslatog koda i klijentske tajne , vraća klijentskoj aplikaciji potrebne tokene.
* Klijentska aplikacija pomocu tokena sada može pristupati zaštićenim resursima, kao i zahtevanjem dodatnih informacija o klijentu , gađajuci /userInfo metodu na strani servera identiteta.

Takođe postoje nekoliko kombinacija hibridnog toka, definisana kroz response\_type parametru prilikom autentikacije i autorizacije.

* **response\_type=code token –** Server identiteta vraća kao http redirekt odgovor kod i token prava pristupa (access\_token)
* **response\_type=code id\_token -**  Server identiteta vraća kao http redirekt odgovor, kod i token identiteta (id\_token)
* **response\_type=code id\_token token -**  Server identiteta vraća kao http redirekt odgovor, kod token identiteta (id\_token) kao i token prava pristupa (access\_token)

**Tok Autorizacionim kodom sa dokazom kljuca za razmenu kodova (Authorization Code Flow with Proof Key for Code Excange - PKCE)**

PKCE OIDC tok je specijalno dizajniran za autentikaciju native i korisnika mobilnih aplikacija.

Dizajniran je kao nadogradnja toka autorizacionim kodom, za aplikacije koje koriste javne klijente i koje ne mogu sigurno čuvati klijentsku tajnu (client\_secret). Ovaj tok se smatra najboljom praksom kada se koriste SPA (single page applications) ili aplikacije za mobilne uređaje.Ključna razlika između standrdnog toka autorizacionim kodom je ta što aplikacije ne moraju da prosleđuju klijentsku tajnu (client\_secret).

PKCE smanjuje bezbednosne rizike za native aplikacije jer klijentske tajne nisu potrebne u izvornom kodu,što ograničava izloženost obrnutom inženjeringu.

**Zahtev za autentikaciju**

GET /oidc/authorize?  
response\_type=code  
&scope=openid%20profile%20email  
&client\_id=s6BhdRkqt3  
&state=af0ifjsldkj  
&code\_challange= PCBxoCJMdDloUVl1ctjvA6VNbY6fTg1P7PNhymbydM  
$ code\_challenge\_method=S256  
&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1  
Host: server.example.com

Kao što možemo videti zahtev je vrlo sličan toku autorizacionim kodom tako da cemo objasniti samo parametre specifične za PKCE tok.

* **code\_challange-** klijentska aplikacija na svojoj strani generiše random **code\_verifier,** zatim se taj string hešira i ta vrednost se salje kao code\_challange
* **code\_challange\_method-** korišćena hash metoda za heširanje code\_verifier (uglavnom se koristi SHA256)

**Uspešan odgovor nakon potvrde identiteta**

Odgovor je isti kao kod toka autorizacionim kodom. Metoda za autorizaciju vratiti HTTP redirekt (Sa Kodom Autorizacije), klijentskoj aplikaciji.

HTTP/1.1 302 Found  
Location: <https://client.example.org/cb?>  
code=SplxlOBeZQQYbYS6WxSbIA  
&state=af0ifjsld

Kao rezultat , pretraživač šalje GET zahtev klijentskoj aplikaciji u sledećoj formi:

GET <https://client.example.org/cb?>  
code=SplxlOBeZQQYbYS6WxSbIA  
&state=af0ifjsldkj

**Zahtev za dobijanje tokena**

POST /oidc/token HTTP/1.1  
Host: server.example.com  
Content-Type: application/x-www-form-urlencoded  
grant\_type=authorization\_code&  
code=SplxlOBeZQQYbYS6WxSbIA&  
redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb&  
client\_id=s6BhdRkqt3&  
code\_verifier=shekrslsnj4kfdksfksk

Kao što možemo videti , sada u zahtevu za dobijanje tokena umesto klijentske tajne (client\_secret) , klijentska aplikacija šalje izvorni string code\_verifier.

Sada server identiteta na osnovu dobijenih code\_challenge i code\_challenge\_method parametra iz predhodnog zahteva može primeniti heširanje istom metodom nad code\_verifier parametru i uporediti da li se nastala heširana vrednost slaže sa code\_challange parametrom, i time potvrdi klijentski zahtev i izda tokene.

Izdavanje tokena i dalji koraci su isti kao kod toka autorizacionim kodom.



**Validacija tokena identiteta**

Po OIDC specifikaciji klijenti moraju validirati token identiteta na sledeći način:

* Ako je token identiteta enkriptovan, dekriptovati ga pomoću ključeva i algoritama koje je klijent naveo tokom registracije, odnosno koje je server identiteta (OP) trebalo da koristi za enkriptovanje tokena identiteta.Ako je o enkripciji dogovoreno sa OP stranom u vreme registracije i token identiteta nije šifrovan, klijentska strana (Relayng Party) treba da ga odbije.
* Identifikator izdavaoca tokena identiteta (OP) mora se tačno podudarati sa vrednošću **iss** tvrdnje (claim) unutar tokena identiteta.
* Klijent mora da validira **aud** (audience) tvrdnju unutar tokena identiteta koja treba da sadrži identifikator klijenta kome je token namenjen (**client\_id**). Aud tvrdnja moze da sadrži niz klijenata kome je token namenjen.Token identiteta mora biti odbijen ukoliko ne sadrži identifikator klijentske aplikacije ili ako sadrži identifikatore klijenata koje nisu od poverenja.
* Ako token identiteta sadrži više klijenata kome je namenjen u **aud** tvrdnji, klijentska aplikacija bi trebalo da verifikuje da je **azp** (predstavlja trenutno autorizovanu stranu) tvrdnja takođe prisutna u tokenu identiteta.
* Ako je **azp** (authorized party) tvrdnja prisutna onda klijent treba da verifikuje da je njena vrednost zapravo identifikator klijenta (**client\_id**).
* Ako je token identiteta primljen preko direktne komunikacije između klijenta i servera identiteta (što i jeste slucaj u OIDC tokovima), onda TSL serverska validacija može biti iskorišćena da validira izdavaoca tokena umesto provere potpisa tokena.Klijentska aplikacija mora validirati potpise svih drugih tokena identiteta prema **JWS** (JSON Web Signature) specifikaciji, koristeći algoritme specificirane u headeru tokena identiteta odnosno **alg** tvrdnji. Klijent mora koristiti ključ koji je dobio od izdavača tokena (OP).
* Unutar zaglavlja (headera) tokena identiteta **alg** tvrdnja bi trebalo da bude **RS256** ili algoritam poslat od strane klijentske aplikacije preko **id\_token\_signed\_response\_alg** parametra prilikom registracije sa serverom identiteta.
* Trenutno vreme mora biti pre vremena isteka tokena identiteta definisano u **exp** (expiration) tvrdnji.
* Tvrdnja iat (issued at) u tokenu identiteta predstavlja vreme izdavanja tokena i može se koristiti za odbijanje tokena koji su izdati predaleko od trenutnog vremena. Prihvatljivi opseg je specifičan za klijenta.
* Ako je **nonce** vrednost poslata prilikom zahteva za autentikaciju, onda **nonce** tvrdnja mora biti prisutna u tokenu identiteta , I proverava se da li je ta vrednost ista kao vrednost koju je klijent poslao prilikom autentikacije, uvedeno je radi sprečavanja takozvanih **replay** napada.

**Single Sign Out**

Kao što smo rekli Single Sign On (SSO) je moguce postići oslanjajuci se na neki centralni server identiteta koji nakon uspešne autentikacije pamti stanje sesije u vidu kolačića, i tokom sledece prijave korisnika preko druge aplikacije, server identiteta proverava da li postoji sesija odnosno kolačić koji potvrdjuje identitet korisnika samim tim govori serveru identiteta da je korisnik već izvršio proces autentikacije i dodeljuje mu potrebne tokene.

Sa druge strane Single Sign Out nije tako jednostavno implementirati. Single Sign Out zahteva da server identiteta (server tokena) sadrži i listu aplikacija kojima je izdao tokene i koji protokol će koristiti za slanje zahtveva za odjavu.Korisnik se mozda prijavio u različite aplikacije koristeći različite protokole (WsFed, SAML, OpenIDConnect). Svi ovi protokoli definišu neke pristupe za Single Sign Out, ali većina njih ne rezultira odjavom iz svih aplikacija. Deo problema jeste sto se navedeni protokoli ne mogu osloniti na prisustvo pretraživača, koji sluša takve poruke nakon inicijalne prijave korisnika (na primer mobilne aplikacije koriste pretraživac samo za postizanje Singl Sign On-a ).

Moguće rešenje je proslešivanje svih zahteva preko nekog dodatnog proxy servera koji ima status prijave korisnika, ali to nije bas skalabilno rešenje.

Za situacije gde nije moguće postići odjavu sa svih aplikacija (pre svega kod aplikacija koje ne koriste sve vreme pretraživač u svom radu), najbolja opcija je ne stvoriti utisak da se korisnik odjavljuje iz svih svojih aplikacija odjednom.

**OIDC Sign Out**

Prema OIDC specifikaciji postoje tri metode za postizanje single sign out-a , odnosno za okončanje korisničke sesije.

* **Session Management**
* **Front-channel Logout**
* **Back-channel Logout**

U nastavku ćemo objasniti detaljnije kako funkcionišu ove 3 metode kao I kako klijentska aplikacija RP (relaying party) inicira zahtev za sign out , prema serveru identiteta.

**Klijentska aplikacija inicira logout**

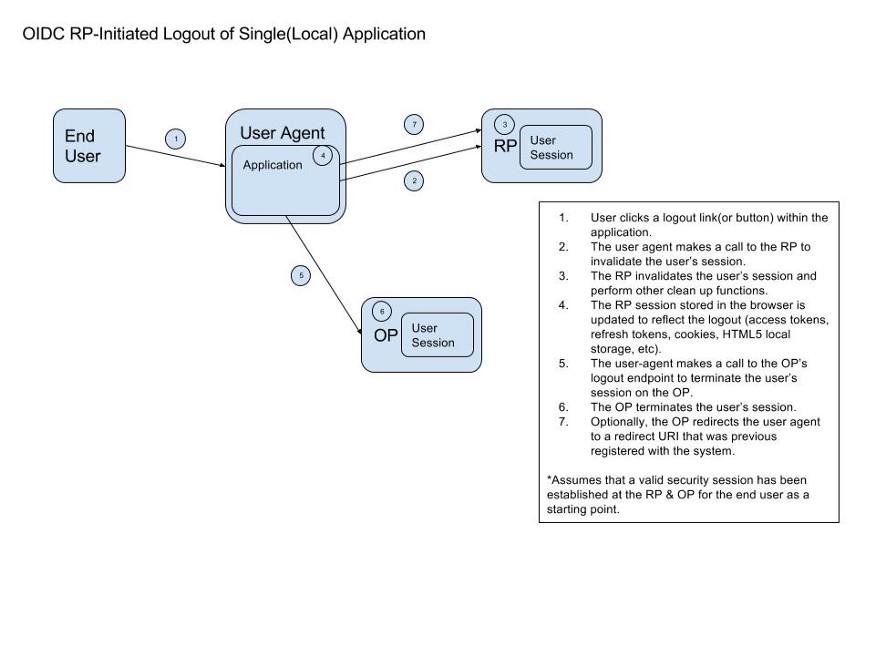
U ovom slučaju klientska aplikacija je odgovorna za brisanje internih tokena u svom skladištu.Nakon što je to završeno , klijentska strana, redirektuje pretraživač na logout metodu unutar servera identiteta (OP).Logout metoda je definisana po OIDC Session Management Specifikaciji I može biti pribavljenja preko OP Discovery Endpoints (Po OIDC specifikaciji unutar Discovery Endpoints se nalazi spisak svih metoda koje pruža server identiteta poput authorize, token, logout, userInfo metoda)

Logout zahtev ka serveru identiteta izgleda ovako:

GET <https://idp.levvel.io/oidc/logout?id_token_hint=previvously_issued_jwt_id_token&>  
post\_logout\_redirect\_uri=https://app1.levvel.io/oidc/logged\_out&  
state=some\_random\_string

Parametri koji se šalju prilikom odjavljivanja, po specifikaciji su sledeći:

* **id\_token\_hint –** Predhodno izdat token identiteta, nije obavezan ali se preporučuje kao nagoveštaj o trenutnoj autentikacionoj sesiji sa klijentom.
* **post\_logout\_redirect\_uri-** Povratna url adresa klijentske aplikacije na koju ce pretraživač biti redirektovan nakon što odjavljivanje bude izvršeno. Ova vrednost mora predhodno biti registrovana u podešavanjima klijenta na strani servera identiteta (OP). Takođe je opcionalni parameter.
* **state-** Vrednost koju klijentska aplikacija koristi za održavanje stanja između zahteva za odjavu I povratnog poziva navedenog kroz post\_logout\_redirect\_uri u cilju sprečavanja CSRF napada.Takođe je opcionalni parameter.



Kada jedna od klijentskih aplikacija izda zahtev za odjavljivanje, korišćenjem jedne od 3 metoda koje definise OIDC , možemo implementirati odjavljivanje ostalih klijentskih aplikacija.

**Session Management**

Upravljanje sesijama je metoda za kontinuirano praćenje statusa prijavljivanja krajnjeg korisnika kod servera identiteta, tako da klijentska aplikacija može da odjavi krajnjeg korisnika koji se već odjavio sa servera identiteta (preko neke druge pouzdane klijentske aplikacije)

Jednostavno rečeno to znači da klijentska aplikacija treba neprestano da pita server identiteta da li je krajnji korisnik jos uvek prijavljen. Ako server identiteta odgovori da korisnik nije prijavljen , onda bi klijentska aplikacija trebala da obriše sve lokalne podatke sesije i tokene povezane sa tim korisnikom.

Postoje dva načina na koja klijentska aplikacija može da proveri status prijavljivanja krajnjeg korisnika.

Jedna od metoda je ponavljanje zahteva za potvrdu identiteta, ali to bi prouzrokovalo previše mrežnog saobraćaja i poremetilo tj odgodilo uslugu servera identiteta.

Kako bi bilo poželjnije proveriti status bez uzrokovanja previše mrežnog saobraćaja, koristi se poboljšana tehnika koja uključuje iframe-ove.Jednom kada se sesija uspostavi , iframe klijentske aplikacije ce idalje ispitivati iframe na strani servera identiteta za status prijavljivanja krajnjeg korisnika.

**Front-channel Logout**

Komunikacija preko Front-channel a se izvršava preko pretraživača. Kada Klijentska aplikacija inicira odjavljivanje , server identiteta preko stranice za logout izrađuje iframe-ove koji se onose na adrese ostalih klijenata koji su registrovani u serveru identiteta. Na ovaj način zahtev za odjavu za svaku klijentsku aplikaciju se javlja paralelno (umesto serijske sesije preusmeravanja da bi se izvršila odjava na svakoj klijentskoj aplikaciji)

**Back-channel Logout**

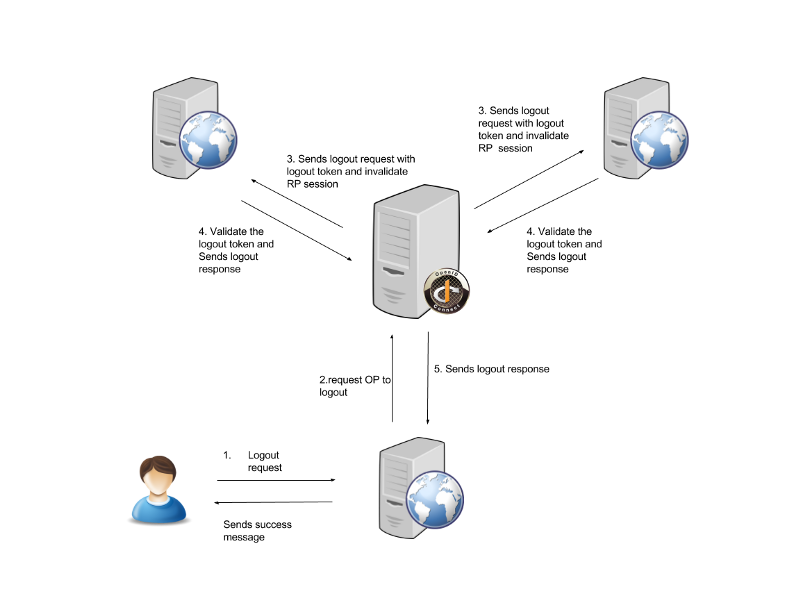
Ovaj mehanizam odjave je mnogo manje “lomljiv“ od ostalih metoda koji koriste front-channel komunikaciju jer ne zavisi od aktivnosti pretraživača krajnjeg korisnika, vec se komunikacija vrši tako sto server identiteta prilikom odjave direktno gadja adrese svih aktivnih klijenata nakon okončanja sesije. Ovo se može žiniti najboljom opcijom za OIDC odjavom. Međutim nije prikladan za svaki scenario jer nameđe neke nove zahteve sistemu.

Kada Klijentska aplikacija ili server identiteta pokrene akciju odjave, server identiteta pronadje sve klijentske aplikacije koje na serveru identiteta dele sesiju istog korisnika.

Server identiteta tada generiše poseban JWT token, koji je jako sličan tokenu identiteta nazvan token za odjavu, koji sadrži relevantne tvrdnje za svaku klijentsku aplikaciju i salje zahteve sa tim tokenima na metodama registrovanim u klijentskim aplikacijama kao krajnje tačke odjave (klijentske adrese za odjavu back-channel metodom moraju da budu registrovane i na strani servera identiteta)

POST /backchannel\_logout HTTP/1.1  
Host: rp.example.org  
Content-Type: application/x-www-form-urlencoded  
logout\_token=eyJhbGci ... .eyJpc3Mi ... .T3BlbklE ...

Po prijemu zahteva klijentska strana validira token za odjavu i ponistava lokalnu korisničku sesiju.



**Prednosti Back-Channel-a**

Ovaj mehanizam odjave je dosta pouzdaniji od ostalih metoda, jer uspeh zahteva za odjavu ne zavisi od aktivnosti sesije korisničkog pretraživača.

**Posledice Back-Channel-a**

Sistemu se nameću novi zahtevi jer se moraju uspostaviti dodatne mrežne veze između servera za back-channel komunikaciju.

Podaci o stanju sesije koji se čuvaju u kolačićima i u HTML5 skladištu, nisu dostupni preko back-channel komunikacije.

U Front-channel metodama, odjava je zapravo jednostavno brisanje kolačića i stanja u HTML5 memoriji.Složenije je u odjavi preko Back-channel komunikacije, jer klijentske aplikacije moraju da primene specifične metode za prekid korisničkih sesija po prijemu zahteva za odjavom.