Protokoli za izdavanje i prezentaciju vjerodajnica u EUDI okviru

Mitrović, Lovre

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:663155

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-14**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 620

PROTOKOLI ZA IZDAVANJE I PREZENTACIJU VJERODAJNICA U EUDI OKVIRU

Lovre Mitrović

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 620

PROTOKOLI ZA IZDAVANJE I PREZENTACIJU VJERODAJNICA U EUDI OKVIRU

Lovre Mitrović

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 620

Pristupnik:	**Lovre Mitrović (0036523529)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	izv. prof. dr. sc. Ante Đerek

Zadatak: **Protokoli za izdavanje i prezentaciju vjerodajnica u EUDI okviru**

Opis zadatka:

U svrhu jačanja interoperabilnosti digitalnog identiteta između zemalja članica EU, Europska komisija je usvojila preporuku kojom poziva zemlje članice da rade na razvoju specifikacija, arhitekture, praksi, smjernica i alata vezanih uz europski digitalni identitet (EUDI). U sklopu diplomskog rada potrebno je istražiti EUDI arhitekturu i referentni okvir, sa naglaskom na protokole za izdavanje i prezentaciju vjerodajnica zasnovane na OpenID4VC standardu. Također, potrebno je istražiti postojeće pokušaje formalne verifikacije spomenutih protokola te sličnih protokola zasnovanih na OpenIdConnect standardu. Na temelju istraživanja, potrebno je ili provesti formalnu analizu jednog od protokola iz OpenID4VC standarda ili izgraditi prototip sustava koji koristi postojeće protokole iz OpenID4VC standards za izdavanje i prezentaciju digitalnih vjerodajnica. Radu je potrebno priložiti izvorni kod razvijenih i korištenih programa, citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 28. lipnja 2024.

Zahvale obitelji, prijateljima i mentoru.

Sadržaj

1. Uvod

U fizičkom svijetu već ustaljeno korištenje raznih vjerodajnica poput osobne iskaznice ili putovnice koja služi kao sredstvo identifikacije i vozačke dozvole koja dokazuje sposobnost osobne o upravljanju motornim vozilom. Kada je nastala potreba za identifikacijom na internetu uočilo se kako je internet zamišljen kao sustav bez identifikacijskog sloja. Stoga su različiti servisi na internetu pribjegli implementiranju vlastitog identifikacijskog sloja. Tako stvorene implementacije uzrokovale su balkanizaciju identiteta na internetu.

U začetku interneta pojavile su se metode za identifikaciju koje su bile izuzetno centralizirane. Razvojem interneta stvoreni su protokoli koji teže decentralizaciji identiteta i stvaranju bolje i jednostavnijeg korisničkog iskustva poput OAuth 2.0 i OpenId Connect-a. Međutim iako su spomenuti protokoli do zaustavili balkanizaciju identiteta i stvorili određenu mjeru decentralizacije nisu uspjeli stvori identitet na internetu koji je u potpunosti u vlasništvu krajnjeg korisnika.

U određenom trenutku državne organizacije su se okrenule digitalizaciji svojih usluga koje zahtijevaju identifikaciju korisnika. Europska Unija je regulacijom eIDAS stvorila okvir za elektroničku identifikaciju fizičkih i pravnih osoba za elektroničke usluge na području Europske Unije. Iako je eIDAS regulacija bila značajni iskorak, koji je dodatno ubrzao digitalizaciju i povećao dostupnost državnih usluga, ona nije bila bez mana.

EUDI je novi okvir koji nastoji proširiti benefite eIDAS regulacije u privatni sektor i pružiti suverenitet nad njihovim identitetom. Navedeni cilj se nastoji ostvariti korištenjem novim skupom protokola i specifikacija o kojima će biti riječ u ovom radu.

2. Pozadina

U odjeljku 2.1. bit će dane rječničke definicije ključnih pojmova za razumijevanje tematike. Zatim će biti dan povijesni pregled razvoja identiteta na internetu 2.2. Te će na kraju biti predstavljeno trenutno stanje identiteta na internetu 2.3.

2.1. Osnovni pojmovi

Osnovni pojmovi za razumijevanje tematike su identitet, vjerodajnica, autentifikacija i autorizacija koji su opisani u ovom odjeljku.

2.1.1. Identitet

Središnji pojam je pitanje identiteta. Definicija identiteta ovisi o području i kontekstu u kojem se nalazi. Pojam identiteta je razmatran najčešće u kontekstu psihologije. Rječnik *American Psychological Assocition* [1] definira **identitet** kao individualni osjećaj definiran skupom fizičkih, psiholoških i međuljudskih karakteristika koje nisu u potpunosti podijeljene s drugom osobom i kao niz pripadnosti i društvenih uloga. Nadalje Kim Cameron [2] je definirao digitalni identitet kao skup tvrdnji jednog digitalnog subjekta o sebi ili o drugom subjektu. Christopher Allen je u svom članku o digitalnom identitetu [3] identitet definirao kao reprezentaciju entiteta koja može sadržavati tvrdnje i identifikatore. Tvrdnje su izjave o entitetu mogu biti činjenice ili mišljenja dok su identifikatori oznake koje unikatno određuju identitet.

2.1.2. Vjerodajnica

Sljedeći pojam od interesa je vjerodajnica. Prema *Marriam-Webster* rječniku [4] **vjerodajnica** je svjedočanstvo ili certificirani dokument koji pokazuje da je osoba ovlaštena ili ima pravo na izvršavanje neke dužnosti te se kao primjer navodi diploma. Nužno je

primijetiti da u kontekstu vjerodajnice postoje tri entiteta: **izdavač** to jest onaj koji izdaje vjerodajnicu, **posjednik** to jest onaj koji prima vjerodajnicu i **provjeravatelj** kojem će posjednik predočiti vjerodajnicu kako bi mu provjeravatelj dozvolio određena prava. Dodatno postoji i entitet **subjekt** a to je onaj na kojega se vjerodajnica odnosi. Najčešće su subjekt i posjednik isti entitet, ali ne moraju biti. Bitno je naglasiti da između posjednika i provjeravatelja ne postoji uspostavljeno povjerenje već je povjerenje uspostavljeno samo između izdavača i provjeravatelja.

Praktični primjer je provjera vozačke dozvole prilikom policijske kontrole. Policajac zaustavlja osobu koja upravlja vozilom. Policajac se treba uvjeriti da je ta osoba ovlaštena upravljati vozilom stoga on ima ulogu provjeravatelja. Vozač kako bi uvjerio policajca predočava vozačku iskaznicu u kojoj je navedeno da ta osoba ima pravo upravljanja vozilom. Vozačka iskaznica ima ulogu vjerodajnica te je ona izdana os strane države koja u ovom kontekstu ima ulogu izdavača. Pošto policajac vjeruje državi on se na osnovu te vjerodajnice uvjerio da navedena osoba ima pravo upravljanja vozilom.

Bitno je naglasiti da se termin vjerodajnica *(eng. credential)* u računarnoj znanosti koristi i za korisnički unos informacija koje identificiraju korisnika poput para korisničkog imena i zaporke.

U sklopu ovog rada definirao bih identitet kao skup tvrdnji koje jedinstveno definiraju neki entitet, a vjerodajnicu kao skup tvrdnji koji se odnosi na nekog subjekta te ispravnost navedenih tvrdnji garantira izdavač vjerodajnice. Ispreplitanje navedena dva pojma je vidljivo jer oba sadržavaju skup nekih tvrdnji.

2.1.3. Autentifikacija

Microsoftov *Computer Dictionary* definira **autentifikaciju** kao proces u kojem sustav provjerava istinitost informacija koje je korisnik unio prilikom prijave. Gdje se korisničko ime i zaporka uspoređuju s listom autoriziranih korisnika [5]. *Economic Times* pruža općenitiju definiciju autentifikacije kao proces prepoznavanja korisničkog identiteta [6]. Razvijeno je više metoda za prepoznavanje identiteta, a jedna od najprimitivnijih je već spomenuta metoda pomoću para korisničkog imena i zaporke.

2.1.4. Autorizacija

Uz pojam autentifikacije često se pojavljuje i pojam autorizacije. Microsoftov *Computer Dictionary* definira **autorizaciju** kao pravo dano korisniku da koristi sustav i podatke pohranjene na njemu [5]. Odnosno kada je korisnik autentificiran i sustav se uvjerio u njegov identitet potrebno je utvrditi imali taj korisnik pravo pristupanja resursu koji je zatražio.

2.2. Povijesni razvoj identiteta

U ovom odjeljku dan je pregled povijesnog razvoja identiteta na internetu. Christopher Allen je u svom članku *The Path to Self-Sovereign Identity* [3] dokumentirao razvoj paradigme identiteta kroz četiri faze: centralizirani identitet, federalni identitet, identitet usmjeren na korisnika i samostalni suvereni identitet.

2.2.1. Centralizirani identitet

Razvoj identiteta na internetu je započeo s centralnim autoritetima koji su bili izdavači i autentifikatori digitalnog identiteta. Organizacija IANA određuje valjanost IP adresa dok ICANN upravlja domenama. Počevši od 1995. certifikacijski autoriteti (skraćeno CA) upravljaju certifikatima. Uz to neki centralni autoriteti identiteta uspostavljaju hijerarhije predajući dio obaveza i odgovornosti na niže grane. Problem s centralnim autoritetima digitalnog svijeta je isti kao i u fizičkom svijetu: korisnici su prepušteni jednom autoritetu koji ih može odbiti ili čak potvrditi lažni identitet. Također identiteti korisnika prema stranicama su postajali sve balkaniziraniji kako je broj stranica rastao.

2.2.2. Federalni identitet

Kako bi se riješio problem balkaniziranosti identiteta stvaraju se federacije. Dok se korisnik kreće sa stanice na stranicu njegov identitet ostaje dok pojedina stranica zadržava autoritet. *Microsoft Passport* je jedan od prvih *single sign-on* servisa (skraćeno SSO) razvijen u ovoj fazi. Iako ovakav pristup rješava rascjepkanost identiteta korisnika prema internetskim stranicama i dalje ostaje problem centraliziranosti.

2.2.3. Identitet usmjeren na korisnika

Treća faza je obilježena razvojem protokola poput OpenID (2004.), OAuth 2.0 (2010.) i OpenID Connecta (2014.) koji su omogućili postavili korisnika u centar omogućavajuću veću interoperabilnost i unoseći element korisničkog pristanka. OpenID također nudi mogućnost registriranja vlastitog OpenId koji korisnik može koristiti autonomno, ali takva registracija zahtjeva određeno znanje koje prosječni korisnik ne posjeduje. Stoga će on koristiti Open ID jedne internetske stranice na drugoj. Tako da problem centraliziranosti i dalje nije u potpunosti riješen.

2.2.4. Samostalni suvereni identitet

Identitet usmjeren na korisnika je pretvorio centralne autoritete u interoperabilne federacije dodajući element korisničkog pristanka, ali korisnik i dalje u potpunosti ne posjeduje svoj identitet. Allen napominje kako i dalje ne postoji jedinstvena definicija samostalnog suverenog identiteta, ali predlaže početnu poziciju naglašavajući kako samostalni suvereni identitet uz interoperabilnost i korisnički pristanak daje potpunu kontrolu korisniku nad tim identitetom što implicira da je takav identitet prenosan.

Vidljivo je kako se Allenov opis samostalnog suverenog identiteta djelomično podudara s već poznatim modelom vjerodajnice u fizičkom svijetu onako kako je opisana u 2.1.2. Posjednik uistinu posjeduje takvu vjerodajnicu i ima mogućnost korisničkog pristanka u vidu pokazivanja vjerodajnice. Pitanje interoperabilnosti bi bilo riješeno postavljajući standard na globalnom nivou.

2.3. Sloj identiteta na internetu

Internet je originalno razvijen bez načina da znamo tko se spaja i na što se spaja. S obzirom na to da je taj bitni sloj izostavljen u ranom razvoju u trenutku kada se pojavila potreba za identificiranjem korisnika svaki sustav je za sebe odabrao način na koji će implementirati ovaj nedostajući sloj identiteta što je dovelo do balkanizacije identiteta i nedostatka jedinstvenog načina identificiranja [2]. U nastavku se daje pregleda trenutnih rješenja implementiranja ovog nedostajućeg sloja identiteta te zakoni koje bi jedan ovakav sloj trebao poštovati kako bi bio uspješan.

2.3.1. Trenutno stanje identiteta na internetu

Današnje stanje identiteta na internetu je uvelike centralizirano. Kada korisnik pristupa nekom internetskom servisu najčešće mu pristupa preko URL koji u sebi sadrži domenu koja usmjerava korisnika na IP adresu. Domene dodjeljuje organizacija ICANN preko svojih podorganizacija dok organizacija IANA regulira IP adrese [7]. Ovaj način identificiranja internetskog servisa dolazi još iz prve faze razvoja identiteta: centralizirani identitet opisanom u 2.2.1.

Kada je korisnik pristupio servisu, ukoliko servis koristi TLS, susreće se s još jednim identitetom iz prve faze razvoja identiteta: digitalnim certifikatima. Digitalni certifikati su tehnologija za identificiranje korisnika ili servisa iako danas su najčešće korišteni isključivo za servise. Digitalni certifikati su izdani od strane certifikacijskog autoriteta koji mogu biti javne ili privatne organizacije koje upravljaju infrastrukturom javnih ključeva *(eng. public key infrastructure)*[8].

Korisnici se najčešće identificiraju servisu pomoću para korisničkog imena i zaporke. Pošto svaki servis zasebno vodi evidenciju o korisnicima dolazi do već spomenute balkanizacije identiteta odnosno korisnici su primorani kreirati zasebni identitet na svakom servisu. Danas većina popularnih servisa ima implementiran primjerice OpenId Connect protokol koji omogućava da se korisnik identificira identitetom trećeg servisa. U praksi to znači da će se korisnik identificirati većini servisa koristeći identitet Googlea, Apple ili Facebooka. Vidljivo je da je i ovdje prisutan problem centraliziranosti identiteta. Stvorila se oligarhija u vidu nekolicine kompanije koje kontroliraju identitet. Ovdje opisani primjer pripada trećoj fazi razvoja identiteta: identitet usmjeren na korisnika opisanom u 2.2.3.

2.3.2. Zakoni identiteta

U svom radu *The Laws of Identity* Kim Cameroon na osnovu dotadašnjih uspjeha i neuspjeha implementiranja identitetskog sloja sastavlja sedam zakona koje bi trebao poštovati dobar i kvalitetan sloj identiteta u nekom sustavu na internetu. U nastavku su nabrojani i ukratko objašnjeni Cameronovi zakoni identiteta [2]:

1. **Korisnička kontrola i pristanak**. Kako bi sustav zaživio mora biti jednostavan

i pogodan za korištenje te mu korisnik mora vjerovati. Da bi zaradio povjerenje korisnika sustav ga mora staviti u poziciju kontrole te mu dati pravo izbora i pravilno ga informirati o izborima koje je donio.

2. **Minimalno otkrivanje za danu uporabu**. Potrebno je krenuti od pretpostavke da je proboj sustava uvijek moguć stoga kako bi se smanjila šteta u slučaju proboja sustav treba pohranjivati minimalno informacija za danu uporabu.

3. **Opravdane stranke**. Otkrivanje identitetskih informacija treba biti ograničeno samo za one stanke koje imaju opravdani interes. Korisnik treba biti obaviješten na koji način će se koristiti dane informacije.

4. **Usmjereni identitet**. Univerzalni sloj identiteta treba podržavati usmjereni identitet. Identitet može bi jednosmjeran ili svesmjeran. Primjer svesmjernog identiteta je URL web stranice. Takav identitet je javan i usmjeren prema svima te se ponaša kao svjetionik *(eng. beacon)* signalizirajući svima na kojoj se adresi stranica nalazi. Kada se korisnik spaja sa stranicom on odabire identifikator na kojim će se identificirati navedenoj stranici. Pošto se korisnik identificira samo stranici takav identitet je jednosmjeran.

5. **Pluralizam operatora i tehnologija**. Univerzalni sloj identiteta treba podržavati više različitih tehnologija i suradnju više različitih pružatelja identiteta.

6. **Ljudska integracija**. Kriptografija je riješila sigurnost komunikacijskog kanala između različitih računala, ali još uvijek je problematičan komunikacijski kanal između računala i čovjeka koji je podložan raznim napadima poput *phisinga*. Univerzalni identifikacijski sloj trebao bi imati siguran mehanizam komunikacije čovjeka i računala.

7. **Konzistentno iskustvo između različitih konteksta**. Korisničko iskustvo mora biti konzistentno i jednostavno dok istovremeno postoje različite tehnologije i operatori u pozadini.

2.4. Formalna verifikacija koda

Formalna verifikacija koda je proces matematičke provjere kojom se nastoji provjeriti je li ponašanje sustava, koje je opisano formalnim modelom, zadovoljava dana svojstva također opisana formalnim modelom. Oba modela moraju dijeliti istu semantičku interpretaciju [9]. Formalna verifikacija je izuzetno korisna kako bi se dokaza sigurnost određenih protokola ili kako bi se pronašle sigurnosne ranjivosti u protokolu.

Pošto je ručna provjera svojstava naporan posao razvijeni su alati u kojima specificiramo oba modela i onda računalo pokuša dokazati odnosno opovrgnuti da model sustava zadovoljava određena svojstva. Jedan od takvih alata jest primjerice **Tamarin** o kojemu će biti riječ u nastavku.

2.4.1. Tamarin

Tamarin Prover je snažan alata za simboličko modeliranje i analizu sigurnosnih protokola. Kao ulaz prima model protokola koji specificira ponašanje agenata u protokolu, ponašanje napadača i željena sigurnosna svojstva [10]. S obzirom na to da je problem dokazivanja sigurnosnih svojstava neodlučiv Tamarin se ne mora zaustaviti. Ako se zaustavi i sigurnosna svojstva drže onda Tamarin priloži dokaz međutim ako sigurnosna svojstva ne drže Tamarin konstruira protuprimjer. Model napadača u Tamarin alatu je Dolev-Yao napadač koji kontrolira mrežu na kojoj se razmjenjuju poruke [11]. Tamarin Prover je detaljno dokumentiran u [10].

Tamarin modeli se sastoji od tri glavne komponente

1. Tvrdnji (eng. Terms)
2. Činjenica (eng. Facts)
3. Pravila (eng. Rules) koja izmjenjuju multiset

Ponašanje protokola se modelira koristeći pravila. Svako pravilo se sastoji od tri dijela: lijeva strana ili premise, oznake tranzicije ili akcijskih činjenica i i desna strana ili konkluzije. Stanje Tamarina je multiskup činjenica. Pravilo se izvodi tako da se činjenice specificirane u lijevoj strani uklanjaju iz stanja, a činjenice u desnoj strani dodavaju u stanje. Akcijske činjenice se koriste u specifikaciji sigurnosnih svojstava.

```
rule Register_pk: [ Fr( ltk)] -> [ !Ltk($A,  ltk), !Pk($A, pk( ltk)) ]
```

Slika 2.1. Tamarin Prover pravilo registracije javnog ključa

Činjenice se mogu promatrati kao predikati koji pohranjuju informaciju.

Postoje posebne unaprijed definirane činjenice poput **Fr** koja se nalazi isključivo na lijevoj strani i označava da se radi o tvrdnji koja je svježe generiran poput kriptografsko ključa. Zatim činjenica **In** koja se nalazi isključivo na lijevoj strani i označava da je tvrdnja koja se može promatrati kao poruka primljena i činjenice **Out** koja se nalazi isključivo na desnoj strani i označava da je tvrdnja to jest poruka poslana. Napadač ima kontrolu nad svim porukama poslanim u mrežu pošto se radi o Dolev-Yao modelu napadača.

Tvrdnje mogu biti primjerice varijable ili konstante. Ako se radi o varijablama njihov tip se raspoznaje prema prefiksu [11]:

- **x** označava svježu varijablu poput nasumičnih brojeva ili kriptografskih ključeva
- **$x** označava javnu varijablu poput identifikatora primjerice poznati URL
- **#x** označava vremensku varijablu i koristi se za modeliranje sigurnosnih svojstava
- **x** to jest nepredznačena varijabla je superset varijabli *x* i *$x*

Primjer jednog pravila u Tamarinu vidljiv je na 2.1. Pravilo se zove *Register_pk* i služi za registriranje javnog ključa. U premisi se nalazi činjenica **Fr(ltk)** koja govori agentu da generira nasumičnu vrijednost *ltk* te u multiskup spremi dvije činjenice. Obje činjenice su predznačene sa *!* što označava da se one trajno nalaze u multiskupu. Primjerice ako se u nekom budućem pravilu ove dvije činjenice nađu na lijevoj strani pravila one neće biti uklonjene iz multiseta. Činjenica *!Ltk($A, ltk)* u multiskup sprema vezu između identifikatora agenta i njegovog sveže generiranog privatnog ključa. Dok činjenica *!Pk($A, pk(ltk))* sprema vezu između identifikatora i javnog ključa. Posebna funkcija **pk** označava javni ključ izveden iz prvog argumenta. Tamarin ima uređen niz posebnih kriptografskih funkcija za kreiranje sažetka, potpisivanje, provjeru potpisa, enkripciju i dekripciju bilo simetričnu bilo asimetričnu i druge.

Sigurnosna svojstva se opisuju pomoću lema *(eng. lemma)*. Tamarin koristi svoj-

```
lemma nonce_secret: " All x #i. Secret(x)@i ==> not (Ex #j.  K(x)@j)"
```

Slika 2.2. Tamarin Prover lema tajnosti

stvo traga kako bi izveo zaključak o sigurnosnim svojstvima. Trag je niz akcijskih činjenica. Trag izvođenja protokola nastaje zabilježavanjem akcijskih činjenica tijekom izvođenja pravila. Svojstvo traga opisano je skupom tragova. Svojstvo traga je valjano za protokol ako je valjano za sve moguće tragove koje je proizveo protokol. Tragovi su definirani logikom prvog reda.

Primjer definicije sigurnosnog svojstva je vidljiv na 2.2. U primjeru se definira lema o tajnosti. Lema kaže da za svaki x u svakom trenutku i vrijedi da ako je u trenutku i zabilježena akcijska činjenica *Secret(x)* ne smije postojati trenutak j u kojemu napadač zna x. *K(x)* je posebna akcijska činjenica definirana u Tamarinu i označava da napadač zna tvrdnju x.

3. Pravne regulacije

Iako je ovaj rad rađen iz perspektive računalne znanosti potrebno je dati kratki uvod iz perspektive prava i postojećih regulacija. U nastavku su ukratko opisane regulative eIDAS i njezina nadogradnja EUDI.

3.1. eIDAS

Regulacija eIDAS (electronic Identification, Authentification and Trust Services) je ustanovila okvir za elektroničku identifikaciju fizičkih i pravnih osoba na internetu i povjerljivim servisima (eng. trust services) za elektroničke transakcije na području Europske Unije [12]. Glavni aspekt eIDAS regulacije je uspostaviti međusobno priznate elektroničke identifikacije (skraćeno eID) između različitih članica Europske Unije pod uvjetom da ispunjavaju propisane regulatorne kriterije i da su propisno obavijestili komisiju [13].

eIDAS specificira razne povjerljive servise (eng. trust services) u privatnom sektoru u svrhu poboljšanje povjerenja i efikasnosti u administrativnim poslovnim procesima. Specificirana rješenje uključuju ePotpise (eng. eSignatures), eVremenske Oznake (eng. eTimestamps), kvalificirane web autentifikacijske certifikate (eng. Qualified Web Authentication Certificates), ePečate (eng. eSeals) i usluge elektroničke registrirane dostave (eng. Electronic Registered Delivery Services)[12].

eIDAS regulacija se suočila s određenim izazovima uključujući razilaženje između država članica u razvoju i interoperabilnosti nacionalnih identifikacijskih schema. U srpnju 2020 komisija je otvorila konzultacije kako bi sakupila povratne informacije o razvoju i održavanju eID povjerljivih usluga (eng. thrust services) u EU [13]. Kako bi se riješili postojeći nedostaci uočeni u procesu konzultacija 2021. komisija predlaže

novu regulaciju za uspostavu Europskog digitalnog identiteta (eng. European Digital Identitly skraćeno EUDI).

3.2. EUDI

Okvir Europskog Digitalnog Identiteta (skraćeno EUDI) je nadogradnja prethodno spomenute eIDAS regulacije. Nova regulacija nastoji ispraviti nedostatke eIDAS-a unaprjeđujući efikasnost i proširujući benefite na privatni sektor. Države članice biće obvezane ponuditi fizičkim i pravnim osobama digitalne novčanike koji će biti povezani s njihovim digitalnim identitetima s dokazom ostalih osobnih atributa poput vozačke dozvole, diplome i bankovnih računa [14].

Ciljevi koje nastoji ostvariti EUDI navedeni su u 3.2.1. EUDI novčanik treba biti u stanju razmjenjivati informacije u različitim uvjetima zato su specificirana 4 tipa toka razmotreni u 3.2.2. Također komponente takvog novčanika nabrojane u 3.2.3. trebaju biti u skladu s određenim specifikacijama i protokolima.

3.2.1. Ciljevi EUDI novčanika

Trenutni ciljevi EUDI novčanika,koji mogu biti nadograđeni u budućnosti, naborojani u EUDI ARF-u su[15]:

1. Sigurna identifikacija prema online servisima
2. Omogućavanje potpuno digitalne Europske vozačke dozvole za online i offline scenarije
3. Pristup zdravstvenim podacima u nacionalnom i inozemnom kontekstu poput eRecepata
4. Pohrana edukacijskih vjerodajnica i profesionalnih certifikata
5. Pružanje jače autentifikacije potrošača u sklopu digitalnih financija
6. Pohrana putnih vjerodajnica kako bi međudržavno putovanje bilo jednostavnije

Korisnici EUDI novčanika mogu koristiti novčanik kako bi primali, pohranjivali i prezentirali ateste o sebi uključujući dokazivanje identiteta. Korisnici bi mogli kreirati

digitalne potpise i pečate koristeći novčanik [15]. Atest u kontekstu EUDI novčanika omogućava autentifikaciju atributa. Dok su atributi u ovom kontekstu karakteristike fizičke ili pravne osobe u elektroničkom formatu. Atest kako ga definira EUDI ARF približno odgovara vjerodajnici iz 2.1.2.

3.2.2. Tipovi toka

EUDI novčanik treba podržavati četiri tipa toka [15]:

1. Bliski nadzirani tok
2. Bliski nenadzirani tok
3. Udaljeni tok na različitim uređajima
4. Udaljeni tok na istom uređaju

Prva dva toka se odnose kada su korisnik i druga strana fizički blizu i koriste se protokoli za prijenos na blizinu (NFC, Bluetooth, QR kod) bez korištenja interneta, ali to ne znači da druge funkcije (osim spomenutog transporta) ne zahtijevaju pristup internetu. U nadziranom toku atributi će biti prezentirani čovjeku, ili pod nadzorom čovjeka, koji se ponaša kao oslanjajuća strana (eng. Relaying Party). U kontekstu eIDAS regulacije oslanjajuća strana je fizička ili pravna osoba koja se oslanja na elektroničku identifikaciju. Dok u nenadziranom toku atributi su prezentirani stroju bez ljudskog nadzora [15].

Posljednja dva toka se odnose kada se prijenos podataka odvija preko interneta. U udaljenom toku na različitim uređajima korisnik EUDI novčanika konzumira informacije sa servisa na drugom uređaju to jest na uređaju na kojem se ne nalazi novčanik. Dok kod udaljenog toka na istom uređaju se konzumiraju informacije s istog uređaja na kojem je novčanik [15].

3.2.3. Arhitekturne komponente novčanika

Sljedeće komponente su identificirane ka gradivni blokovi EUDI novčanika koje će biti potrebno implementirati [15]:

1. **Sustav za upravljanje kriptografskim ključevima**. Komponenta koja pohranjuje podatke poput privatnih ključeva.

2. **Protokol za razmjenu atesta**. Protokol koji definira kako prezentirati ateste i identifikacijske podatke.

3. **Protokol za izdavanje**. Protokol koji definira kako će atesti i identifikacijski podaci biti izdani i u kojem formatu.

4. **Model podataka**. Model podataka definira i opisuje elemente te njihovi interakciju.

5. **Scheme atesta i identifikacijskih podataka**. Schema sadržava strukturu i logičku organizaciju podataka koji definiraju svojstva atesta odnosno atribute korisnika.

6. **Formati atesta i identifikacijskih podataka**. Formati se koriste kako bi se predstavile karakteristike i dozvole fizičke ili pravne osobe ili objekta u obliku potpisanih i provjerljivih digitalnih artefakata.

7. **Formati potpisa**. Tehnička implementacija jedne ili više matematičkih metoda u obliku digitalnog artefakta kako bi se dokazala autentičnost i integritet digitalnog dokumenta.

8. **Model povjerenja**. Skup pravila kako bi se osigurala legitimnost komponenti u infrastrukturi EUDI novčanika poput korisničke autentifikacije.

9. **Kriptografski algoritmi i metode**. Algoritmi i metoda za osiguravanje povjerenja i integriteta podataka.

10. **Identifikatori entiteta**. Jedinstveni identifikatori za sve elemente modela podataka.

11. **Provjera valjanosti statusa**. Mehanizmi za objavu i prikupljanje informacije o valjanosti identifikacijskih podataka, atesta, certifikata itd.

EUDI ARF specificira koji protokoli moraju ili mogu biti korišteni za određene komponente ovisno o tipu konfiguracije. Za **protokol za razmjenu atesta** se koristi

OpenID4VP kada se radi o udaljenom toku, a u slučaju bliskog toka koristi se ISO/IEC 18013-5:2021 (mobilna digitalna vozačka). Kod **protokola za izdavanje** zahtjeva se korištenje OPENID4VCI protokola, ali naglašava se kako država članice mogu slobodno uključiti i druge protokole. Kao **model podataka** zahtjeva se onaj specificiran u ISO/IEC 18013-5:2021 i W3C Verfiable Credentials Data Model 1.1. Zahtjevi za **format atesta i identifikacijskih podataka** su JWT, SD-JWT, CBOR i JSON-LD formati. Dok su **formati potpisa** oni u JSOE(JWT) i COSE specifikacijama te LD-Proof specifikacijom [15]. Neki od navedenih protokola bit će detaljnije razmotreni u nastavu.

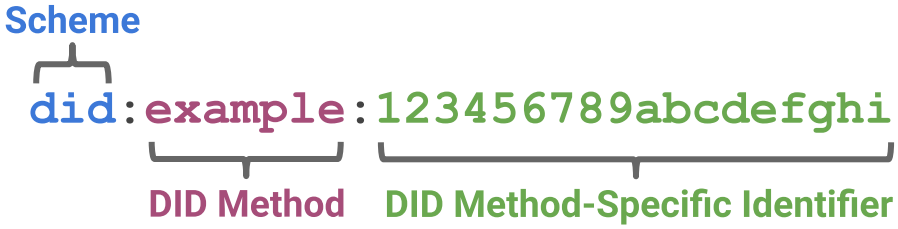
4. Protokoli i specifikacije

4.1. Decentralizirani identifikatori (DID)

Decentralizirani identifikatori (skraćeno DID) je tip identifikatora koji omogućava provjerljive decentralizirane digitalne identitete. Za razliku od federativnih identifikatora DID je dizajniran tako da može biti odvojen od centraliziranih registara, pružatelja identiteta i certificiranih autoriteta. Treće strane mogu biti korištene u otkrivanju informacija vezanih za DID, ali dizajn omogućava da onaj koji kontrolira DID može dokazati kontrolu bez dopuštenja trećih strana. DID je zapravo URI koji asocira DID subjekta sa DID dokumentom omogućavajući interakcije sa subjektom [16].

Svaki DID dokument sadržava kriptografski materijal, verifikacijske metode ili usluge koje pružaju skup mehanizama koji omogućuju DID kontroleru da dokaže kontrolu na DID-om. Servisi omogućavaju interakcije povezane s DID subjektom[16].

Kao što je već rečeno DID je zapravo URI string. Te se sastoji od tri dijela koji su odvojeni dvotočjem: scheme, metode i identifikatora specifičnog za metodu kao što je vidljivo na slici 4.1. Schema je uvijek "did" i ona specificira da se radi o decentraliziranom identifikatoru. Metoda specificira način na koji se DID dereferencira u DID dokument, a treći dio u potpunosti ovisi o metodi. Na slici 4.5. su prikazani primjeri nekih DID metoda.



Slika 4.1. Dijelovi decentraliziranog identifikatora: schema, metoda i identifikator specifičan za metodu

`did:ethr:0xc530503a148babcaca68565cfa576d6f43427a2d`

Slika 4.2. Primjer metode ethr

`did:web:veramo-agent.herokuapp.com`

Slika 4.3. Primjer metode web

`did:key:z6MkpTHR8VNsBxYAAWHut2Geadd9jSwuBV8xRoAnwWsdvktH`

Slika 4.4. Primjer metode key

Slika 4.5. Primjeri korištenja did metoda preuzeti sa [17]

Metoda **did:ethr** pruža skalabilnu identifikacijsku metodu za Ethereum adresu. U pozadini se ova metoda oslanja na tehnologiju Ethereum lanca blokova točnije na pametni ugovor *ethr-did-registry* [17].

S druge strane **did:web** metoda se oslanja na već uveliko rasprostranjen i korišten DNS za razlučivanje DIDa. Kreiranje ovakvog DID je jednostavno. Kako bi se kreirao ovakav DID potrebno je postaviti DID dokument na poznati URL i taj resurs mora biti dostupan HTTPS GET metodom. Sigurnost ove metode proizlazi iz postojeće internetske infrastrukture javnog ključa (PKI) i TLSa [17]. Tako će se primjerice DID `did:web:LovreMitrovic.github.io:did-database:issuer` razlučiti u DID dokument dostupan na `https://lovremitrovic.github.io/did-database/issuer/did.json`.

Također postoji još jednostavnija metoda **did:key** koja ne zahtijeva vanjske alate poput lanca blokova ili DNS-a. Ovaj DID se uvijek razluči u isti DID dokument s obzirom na to da je nepromjenjiv [17]. U prve dvije metode identifikator specifičan za metodu je sadržavao adresu resursa na lancu blokova odnosno na webu dok u ovom slučaju on sadržava javni ključ.

Nadalje DID dokument u koji će se DID razlučiti sadržava podatke o DID subjektu to jest o onome na koga se identitet odnosi te mehanizme poput kriptografskih javnih ključeva koji omogućavaju DID subjektu ili DID delegatu da se autentificira i dokaže svoju povezanost s DIDom. DID dokument može imati jednu ili više reprezentacija primjerice JSON ili JSON-LD [16]. Na slici 4.6. je primjer DID dokumenta koji je dobiven razlučivanjem `did:web:LovreMitrovic.github.io:did-database:issuer`. U dokumentu je vidljiva verifikacijska metoda koja sadržava javni ključ u JWT formatu i DID kontrolera koji je u ovom slučaju i DID subjekt.

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/jws-2020/v1"
  ],
  "id": "did:web:LovreMitrovic.github.io:did-database:issuer",
  "verificationMethod": [
    {
      "id": "did:web:LovreMitrovic.github.io:did-database:issuer",
      "type": "JsonWebKey2020",
      "controller":
        "did:web:LovreMitrovic.github.io:did-database:issuer",
      "publicKeyJwk": {
            "kty": "EC",
            "x": "6v0wx7K7CegMKNzggIlkmok7KLgwbxXScI86emXsTH8",
            "y": "yPBO_uMtUMLTKkkh7NxcwY3JR_fgE6d2VT1N6cxaRp8",
            "crv": "P-256"
            }
      }
  ]
}
```


Slika 4.6. Primjer DID dokumenta

4.2. W3C Verifiable Credentials Data Model 1.1

W3C Verifiable Credentials Data Model 1.1 je specifikacija koja definira model podataka koji će koristiti provjerljive vjerodajnice. **Provjerljiva vjerodajnica** može sadržavati sve informacije kao i fizička vjerodajnica poput informacija o subjektu vjerodajnice, izdavaču, tipu vjerodajnice i tako dalje, ali dodatno sadržava i mehanizme za provjeru kao što su digitalni potpisi. Nadalje posjednik vjerodajnice može generirati **provjerljive prezentacije** i onda podijeliti te provjerljive prezentacije s provjeravateljem kako bi dokazali da posjeduju određene vjerodajnice [18]. *W3C Verifiable Credentials Data Model 1.1* je jedan od protokola koji se koristi u EUDI okviru kao model podataka. U 4.2.1. je opisano kako model izgleda kada se radi o provjerljivoj vjerodajnici dok je u 4.2.2. opisan model u slučaju provjerljive prezentacije. Također u 4.2.3. je opisano kako spomenuti model podataka izgleda u formatu JSON Web Token (skraćeno JWT) koji je jedan od formata korišten za format atesta i identifikacijskih podataka prema EUDI okviru. S obzirom na to da je JWT moguće digitalno potpisati koristeći *JSON Object Signing and Encryption (JOSE)* standard, EUDI okvir ga dopušta kao format potpisa.

4.2.1. Provjerljiva vjerodajnica

Vjerodajnica je skup jedne ili više tvrdnji o istom entitetu. Vjerodajnice mogu sadržavati identifikatore i meta podatke koji opisuju svojstva vjerodajnice poput izdavača, roka trajanja, reprezentativne slike, javnog ključa u svrhu provjeravanja tvrdnji, mehanizma poništavanja i tako dalje. Provjerljiva vjerodajnica je skup tvrdnji i metapodataka koji kriptografski dokazuju tko ju je izdao [18]. U nastavku su opisana osnovna svojstva koja se pronalaze u provjerljivim vjerodajnicama [18]. Primjer provjerljive vjerodajnice je vidljiv na slici 4.7.

1. **@context** svojstvo mora biti uređeni skup URI vrijednosti gdje je prva vrijednost skupa uvijek `https://www.w3.org/2018/credentials/v1`. Svrha ovog svojstva je uspostava konteksta između dva sustava koja komuniciraju.

2. **id**svojstvo se koristi kao identifikator koji jedinstveno identificira neki entitet. Vrijednost ovog polja je URI ili DID.

3. **type** svojstvo se koristi kako bi se provjerili je li dobivena vjerodajnica ili prezentacija odgovarajuća. Vrijednost ovog svojstva je lista tipova koji trebaju biti prisutna u nekom od URI iz svojstva @context. Svaka provjerljiva vjerodajnica treba sadržavati barem tip "VerifiableCredential". Informacije o tim tipovima su dostupne na `https://www.w3.org/2018/credentials/v1`.

4. **credentialSubject** svojstvo sadržava tvrdnje o jednom ili više subjekata. Vrijednost ovog svojstva je skup objekata gdje koji sadrže svojstva o subjektu ili samoj vjerodajnici. Svaki objekt može sadržavati i id svojstvo.

5. **issuer** svojstvo mora biti URI ili objekt koji sadržava id polje koji identificira izdavača vjerodajnice

6. **issuanceData** svojstvo označava trenutak od kojega je vjerodajnica važeća

7. **expirationDate** svojstvo označava trenutak do kojega je vjerodajnica važeća

8. **proof** svojstvo sadržava jedan ili više kriptografskih dokaza koje potvrđuju autentičnost i integritet vjerodajnice ili prezentacije.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "id": "http://example.edu/credentials/3732",
  "type": [
    "VerifiableCredential",
    "UniversityDegreeCredential"
  ],
  "issuer": "https://example.edu/issuers/14",
  "issuanceDate": "2010-01-01T19:23:24Z",
  "credentialSubject": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "degree": {
      "type": "BachelorDegree",
      "name": "Bachelor of Science and Arts"
    }
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2022-02-25T14:58:43Z",
    "verificationMethod": "https://example.edu/issuers/14#key-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "
     z4kWncP1KLByEaSU3oaijUNk8GPGCCgntz8q4Gk55QuMwQe1dsWgSmf7
RsRNYgfJUChdSV22khsfpBsX7ub14aYbe"
  }
}
```


Slika 4.7. Primjer W3C provjerljive vjerodajnice preuzet sa [18]

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  "id": "urn:uuid:3978344f-8596-4c3a-a978-8fcaba3903c5",
  "type": ["VerifiablePresentation", "CredentialManagerPresentation"],
  "verifiableCredential": [{  }],
  "proof": [{  }]
}
```

Slika 4.8. Primjer strukture W3C provjerljive prezentacije preuzet sa [18]

4.2.2. Provjerljiva prezentacija

Provjerljiva prezentacija izražava podatke iz jedne ili više provjerljivih vjerodajnica i sastavlja ih u jednu prezentaciju na način da je autorstvo vjerodajnice provjerljivo [18].

1. **@context** isto kao i kod vjerodajnica
2. **id** isto kao i kod vjerodajnica
3. **type** isto kao i kod vjerodajnica uz iznimku da mora sadržavati barem tip "VerifiablePresentation".
4. **verifiableCredential** svojstvo se konstruira od jedne ili više provjerljivih vjerodajnica ili podataka deriviranih iz provjerljivih vjerodajnica
5. **holder** svojstvo je URI entiteta koji generira prezentaciju

4.2.3. Korištenje JWT formata

Vjerodanica i prezentacija mogu biti upakirane u *JSON Web Token (JWT)* formatu i potpisane koristeći *JSON Object Signing and Encryption (JOSE)*. U tom slučaju JWT treba sadržavati svojstvo *vc* čija vrijednost je provjerljiva vjerodajnica u JSON formatu odnosno svojstvo *vp* čija vrijednost je provjerljiva prezentacija u JSON formatu. Ako je prisutan Json Web Signature (skraćeno JWS) moguće je ispustiti svojstvo *proof* te se podrazumijeva da se potpis se odnosi na izdavača u slučaju provjerljive vjerodajnice odnosno na posjednika u slučaju provjerljive prezentacije. Ukoliko nije prisutan JWS svojstvo*proof* mora biti prisutno i može se koristiti za kompleksije dokaze [18].

JOSE zaglavlje mora slijediti sljedeća pravila:

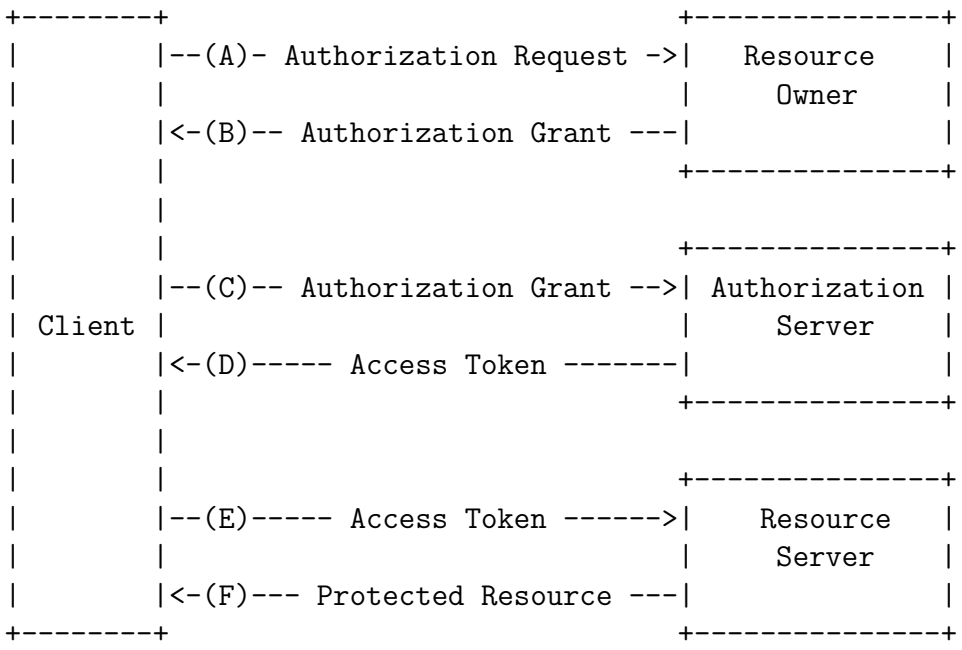
1. **alg** svojstvo mora biti postavljeno na algoritam za potpisivanje, a ako je svojstvo *proof* postojano onda se svojstvo *alg* postavlja na *none*
2. **kid** svojstvo može biti prisutno ako postoji više ključeva asociranih s izdavačem
3. **typ** svojstvo mora biti postavljeno na vrijednost "JWT"

U svrhu unazadne kompatibilnosti s JWT procesorima sljedeća svojstva moraju biti korištena:

- **exp** svojstvo mora odgovarati svojstvu *expirationDate* enkodirano u UNIX timestamp formatu
- **iss** svojstvo mora odgovarati svojstvu *issuer* kada se radi o provjerljivoj vjerodajnici odnosno svojstvu *holder* kada se radi o provjerljivoj prezentaciji
- **nbf** svojstvo mora odgovara svojstvu *issuanceDate* enkodirano u UNIX timestamp formatu
- **jti** svojstvo mora odgovara svojstvu *id* provjerljive vjerodajnice odnosno provjerljive prezentacije
- **jti** svojstvo mora odgovara svojstvu *id* sadržanog u svojstvu *credentialSubject*
- **aud** svojstvo mora odgovara namijenjenoj publici provjerljive prezentacije to jest namijenjenom provjeravatelju

4.3. OAuth 2.0

OAuth 2.0 je autorizacijski okvir koji omogućuje aplikacija treće strane da dobiju pristup HTTP servisu u ime vlasnika resursa (eng. resource owner) ili s njegovim odobrenjem u svoje ime [19]. Ovaj autorizacijski okvir je razvijen u trećoj fazi identiteta na internetu: identitet usmjeren na korisnika. Okvir omogućuje da se identitet s jednog servisa koristi na više različitih servisa nakon korisničkog pristanka na dijeljenje informacija. OAuth 2.0 definira četiri uloge [19]:



Slika 4.9. Apstraktni tok OAuth 2.0 protokola preuzet sa [19]

- **Vlasnik resursa** *(eng. resource owner)* je entitet koji daje pravo pristupa zaštićenom resursu
- **Server s resursom** *(eng. resource server)* je server na kojemu se nalazi zaštićeni resurs i koji na osnovu pristupnog tokena dozvoljava pristup resursu.
- **Klijent** *(eng. client)* je aplikacija koja će pristupati zaštićenom resursu s odobrenjem vlasnika resursa
- **Autorizacijski server** *(eng. authorization server)* je server koji izdaje pristupne tokene klijentu nakon autentifikacije vlasnika resursa i dobivanjem njegove autorizacije.

Apstrakti tok protokola je prikazan na 4.9. Tok započinje tako da klijent zatraži autorizaciju od vlasnika resursa. Autorizacijski zahtjev *(eng. Authorization request)* može biti zatražen neposredno ili posredno preko autorizacijskog servera. Nakon što je zahtjev odobren klijent dobiva autorizacijsko odobrenje *Authorization Grant* koje će dalje biti proslijeđeno autorizacijskom serveru u zamjenu za pristupni token *(eng. Access Token)* ako je odobrenje valjano [19]. Pomoću pristupnog tokena će klijent od servera s resursom zatražiti zaštićeni resurs.

Autorizacijsko odobrenje je vjerodajnica koja je reprezentacija autorizacije vlas-

nika resursa. OAuth 2.0 definira četiri oblika autorizacijskog odobrenja: autorizacijski kod, implicitno autorizacijsko odobrenje, vjerodajnica vlasnika resursa (na primjer korisničko ime i zaporka) i klijentska vjerodajnica. Uz četiri navedena oblika definirani su i mehanizmi za dodavanje dodatni oblika [19]. U nastavku se razmatra samo autorizacijsko odobrenje u obliku **autorizacijskog koda** jer je to oblik na koji se nadograđuju OID4VC protokoli.

Protokol definira dvije krajnje HTTP točke *(eng. HTTP endpoints)* na autorizacijskom serveru [19]:

- **Autorizacijska krajnja točka** se koristi od strane klijenta za zatraživanje autorizacijskog odobrenja i na njoj se traži autentifikacija vlasnika resursa.
- **Krajnja točka za token** se koristi od strane klijenta za razmjenu autorizacijskog odobrenja za pristupni token. Na ovoj točki se od klijenta može tražiti autentifikacija.

Također definirana je jedna krajnja HTTP točka na klijentu:

- **Preusmjerena krajnja točka** se koristi od strane autorizacijskog servera te se na nju vraća odgovor koji sadrži autorizacijske vjerodajnice klijenta.

4.3.1. Autorizacijska krajnja točka

Na autorizacijskoj krajnjoj točki se odvija interakcija s vlasnikom resursa kojeg je potrebno autentificirati. Način na koji autorizacijski server autentificira vlasnika resursa je izvan OAuth 2.0 specifikacije. Moguće je koristiti primjerice par korisničkog imena i zaporke ili kolačiće. Također način na koji klijent saznaje lokaciju krajnje točke je izvan OAuth 2.0 specifikacije. Autorizacijska krajnja točka mora podržavati slanje zahtjeva HTTP GET metodom, ali može podržavati i HTTP POST metodu. Također korištenje TLS je obavezno s obzirom na to da se prenose povjerljivi podaci. Parametri trebaju biti enkodirani u *application/x-www-form-urlencoded* formatu [19].

Kada se klijent šalje autorizacijski zahtjev šalju se sljedeći parametri [19]:

- **response_type** treba biti postavljen na vrijednost *code* jer se radi o autorizacijskom kodu

- **client_id** je unikatni identifikator klijenta
- **redirect_uri** je opcionalni parametar koji treba biti postavljen na URI na koji će se klijent preusmjeriti nakon autentifikacije vlasnika resursa
- **scope** je opcionalni parametar koji specificira opseg i mogućnosti tokena koji će u budućnosti biti izdan
- **state** je preporučeni parametar koji klijent koristi da održava stanje između zahtjeva i povratne veze *(eng. callback)*

Nakon uspješnog procesuiranja autorizacijskog zahtjeva od strane klijenta i autentifikacije vlasnika resursa klijent dobiva HTTP Redirect odgovor koji preusmjerava korisnika na *redirect_uri* skupa sa HTTP query parametrom *code*. Vrijednost tog parametra je generirana na serveru i navedeni *code* ne smije biti korišten više od jednom. Također preporučljivo je da istekne nakon 10 minuta [19].

4.3.2. Krajnja točka za izdavanje tokena

Nakon što klijent primi *code* potrebno ga je zamijeniti za pristupni token. Pristupni token se može dobiti i na osnovu *osvježavajućeg tokena* ,ali to nije tema ovoga rada. Isto kao i u prethodnom koraku lokacija ove krajnje točke nije specificirana dokumentacijom, parametri trebaju biti formatirani u *application/x-www-form-urlencoded* formatu i korištenje TLS je obavezno. Za razliku od prethodnog koraka ovdje treba koristiti isključivo HTTP POST metodu. Također na ovoj krajnjoj točki potrebno je autentificirati klijenta ako se radi o povjerljivom klijentu *(eng. confidential client)*. Autentifikacija klijenta se tipično radi putem zaporke ili para javnog i privatnog ključa. Ako se radi o drugom tipu klijenta tzv. javnom klijentu *(eng. public client)* autentifikacija nije nužna [19].

Prilikom zahtijevanja tokena zahtjev treba imati sljedeće parametre [19]:

- **grant_type** koji treba biti postavljen na vrijednost *authorization_code* s obzirom na to da se o autorizacijsko odobrenju *(eng. Authorization Grant)*
- **code** treba biti postavljen na primljenu vrijednost *code* koja je dobivena u prethodnom koraku

- **redirect_uri** je obavezna vrijednost ako se nalazio u autorizacijskom zahtjevu te vrijednost parametra treba biti identična kao i u autorizacijskom zahtjevu
- **client_id** je obavezna vrijednost ako klijent nije autentificiran odnosno ukoliko se radi o javnom klijentu

Ako je zahtjev ispravan i autoriziran onda se u tijelu odgovora šalju sljedeće parametre[19]:

- **access_token** je pristupni token izdan od strane autorizacijskog servera
- **token_type** je tip tokena
- **expires_in** je preporučeni parametar koja treba koliko sekundi token vrijedi od trenutka izdavanja
- **refresh_token** je opcionalni parametar čija vrijednost odgovara osvježavajućem tokenu
- **scope** je opcionalni parametar koji specificira opseg i mogućnosti tokena koji je izdan te treba biti identičan vrijednosti u zahtjevu

Također HTTP zaglavlja *Cache-Control* odnosno *Pragma* trebaju biti postavljeni na *no-store* odnosno na **no-cache** s obzirom na to da se prenose tokeni.

4.3.3. Pristup zaštićenom resursu

Uz pomoć pristupnog tokena klijent će pristupiti zaštićenom resursu na serveru sa zaštićenim resursom. Server sa zaštićenim resursom mora provjeriti valjanost tokena i njegov opseg. Metoda pristupa zaštićenom resursu je izvan specifikacije protokola, ali najčešće se radi koristeći HTTP Authorization polje zaglavlja HTTP zahtjeva [19].

4.4. OID Connect

OAuth 2.0 protokol je uspostavio okvir za autorizaciju međutim nedostaje mu sloj identiteta. **OpenId Connect 1.0** (skraćeno. OIDC) je jednostavni sloj identiteta iznad OAuth 2.0 protokola. Omogućava klijentu da potvrdi svoj identitet krajnjem korisniku na osnovu autentifikacije koju obavlja autorizacijski server te pruža način

da se dobave informacije o krajnjem korisniku u interoperabilnom način i REST-like načinu [20].

Korištenje ove ekstenzije zahtjeva klijent postavljajući *scope* parametar na vrijednost *openid* u autorizacijskom zahtjevu. Informacije o autentifikaciji i identitetu bit će dostupne u JWT tokenu zvanom *ID token*. Autorizacijski server iz OAuth 2.0 specifikacije u okviru OpenID Connect ima ulogu OpenID pružatelja *(eng. OpenID Provider (OP))*, a klijent ima ulogu oslanjajuće strane *(Relaying Party (RP))*. Specifikacija pretpostavlja da RP zna konfiguracijske informacije o OP-u poput autorizacijske krajnje točke *(eng. Authorization endpoint)* i krajnje točke za izdavanje tokena *(eng. Token endpoint)* [20].

OpenID Connect je moguće koristiti sa svim OAuth 2.0 tokovima autorizacijskog odobrenja, ali u nastavku će biti riječ samo o slučaju kada se koristi tok autorizacijsko odobrenje u obliku autorizacijskog koda. Koraci toka autorizacijskog koda su[20]:

1. Klijent pripremi autentifikacijski zahtjev sa željenim parametrima
2. Klijent pošalje zahtjev autorizacijskom serveru
3. Autorizacijski server autentificira krajnjeg korisnika i dobije njegov pristanak
4. Autorizacijski server preusmjeri krajnjeg korisnika na klijenta zajedno sa autorizacijskim kodom
5. KLijent pošalje zahtjev s autorizacijskim kodom na krajnju točku za izdavanje tokena
6. Klijent primi odgovor s ID tokenom i pristupnim tokenom
7. Klijent provjeri valjanost ID tokena

4.4.1. ID token

Primarna nadogradnja OpenID Connect protokola je ID token struktura podataka koja omogućuje krajnjem korisniku da se autentificira. ID token kreira autorizacijski server te on sadrži tvrdnje o krajnjem korisniku koje je zatražio klijent. ID token je reprezentiran u JWT formatu i mora biti potpisan i opcionalno enkriptiran koristeći *JSON Web*

Signature (JWS) odnosno *JSON Web Encryption (JWE)* standarde. Sljedeće tvrdnje su obavezne za ID tokene u svim OAuth 2.0 tokovima:

- **iss** je identifikator izdavača u obliku URL-a
- **sub** je identifikator subjekta koji treba biti lokalno unikatan
- **aud** je identifikator publike za koju je ID token namijenjen i on mora sadržavati OAuth 2.0 *client_id* vrijednost od RP te može sadržavati identifikator za ostalu publiku
- **exp** je trenutak isteka valjanosti ID tokena u formatu JSON broja koji predstavlja broj sekundi od 1970-01-01T0:0:0Z
- **iat** je trenutak izdavanja ID tokena u formatu JSON broja koji predstavlja broj sekundi od 1970-01-01T0:0:0Z

4.4.2. Autorizacijska krajnja točka

Autorizacijska krajnja točka mora podržavati GET metodu, ali za razliku od OAuth 2.0 u ovom okviru podržavanje POST metode je obavezno. Ako se koristi GET metoda parametri se prosljeđuju koristeći URI query string, a ako se koristi POST metoda parametri se prosljeđuju u tijelu HTTP zahtjeva koristeći format *application/x-www-form-urlencoded*. Obavezni parametri zahtjeva su:

- **response_type** isto kao i u 4.3.1.
- **client_id** isto kao i u 4.3.1.
- **redirect_uri** koji je bio opcionalan u OAuth 2.0 je obavezan u kontekstu OIDC te se mora poklapati s nekim od redirect URI vrijednosti koje je klijent prethodno registrirao na OP-u. U toku autorizacijskog koda URI treba koristiti https shemu, ali može koristiti i http shemu ako se radi o povjerljivom klijentu kako ga definira OAuth 2.0 i ako OP dopušta korištenje http scheme
- **scope** koji je bio opcionalan u OAuth 2.0 je obavezan u kontekstu OIDC treba biti postavljen na vrijednost *openid*

- **state** isto kao i u 4.3.1.

Postoje još i brojni opcionalni parametri, ali najbitnije je naglasiti parametar **nounce** koji treba biti slučajno generiran string. Njegova svrha je uspostava veze između ID tokena i klijentske sesije.

Nakon što je provjerena ispravnost primljenog zahtjeva autorizacijski server treba autentificirati krajnjeg korisnika. Način autentifikacije nije specificiran OIDC specifikacijom, ali najčešće se koriste zaporke ili kolačići. Nakon autentifikacije potrebno je dobiti pristanak krajnjeg korisnika. Na kraju korisnik je preusmjeren na Redirect URI skupa sa autorizacijskim kodom isto kao što je opisano u 4.3.1.

4.4.3. Krajnja točka za izdavanje tokena

Na krajnjoj točki za izdavanje tokena, klijent dobiva ID token, pristupni token i token za osvježavanje. Ako se radi o povjerljivom klijentu njega je potrebno autentificirati, a parametri zahtjeva su isti kao i u 4.3.2.

Odgovor uspješnog zahtjeva treba biti u formatu *application/json*. U odgovoru se šalju isti parametri kao i u 4.3.2., ali *token_type* treba biti postavljen na *Bearer* ako nije drugačije ispregovarano s klijentom. Serveri po specifikaciji trebaju podržavati *Bearer* tokene, a uporaba ostalih tipova je izvan specifikacije.

4.5. OpenID4VC

OpenID for Verifiable Credentials se sastoji od tri specifikacije [21]:

- **OpenID for Verifiable Credentials Issuance (OID4VCI)** definira API iznad OAuth protokola za izdavanje provjerljivih vjerodajnica
- **OpenID for Verifiable Presentations** je ekstenzija OpenID Connect protokola koja omogućava provjeravatelju da zatraži i primi provjerljive prezentacije
- **SIOP v2** je protokol koji omogućava krajnjem korisniku da koristi vlastiti OpenID poslužitelja (OP) kojeg on kontrolira tzv. *Self-Issued OpenID Provider (SIOP)*

4.6. OpenId for Verifiable Credential Issuance

OpenID for Verifiable Credential Issuance (OID4VCI) je specifikacija koja definira API iznad protokola OAuth 2.0 za izdavanje provjerljivih vjerodajnica. Provjerljive vjerodajnice mogu biti u bilo kojem formatu [22]. Protokol će implementirati izdavač vjerodajnica i novčanik u kojem će se vjerodajnice pohraniti. Izdavač odgovara autorizacijskom serveru i serveru sa zaštićenim resursom iz OAuth 2.0 specifikacije, dok klijentu odgovara aplikacija novčanika. Vlasniku resursa iz OAuth 2.0 specifikacije odgovara krajnji korisnik.

API definira šest dodatnih krajnjih točaka [22]:

- Krajnja točka za izdavanje vjerodajnica
- Opcionalna krajnja točka za izdavanje skupa vjerodajnica
- Opcionalna krajnja točka za odgođeno izdavanje vjerodajnica
- Opcionalni mehanizam u kojem izdavač šalje ponudu vjerodajnice *(eng. Credential offer)* novčaniku da bi novčanik pokrenuo tok
- Opcionalni mehanizam u kojem izdavač prima obavijesti od novčanika o statusu izdane vjerodajnice
- Mehanizam koji omogućuje izdavaču objavu svojih metapodataka

Metapodatke će izdavač objaviti na krajnjoj točki čija putanja dobije spajanjem svoga identifikator sa stringom `/.well-known/openid-credential-issuer`. Metapodaci trebaju biti objavljeni u obliku JSON dokumenta te pristup krajnjoj točki treba biti osiguran TLS-om.

OID4VCI specifica dva toka: tok autorizacijskog koda *(eng. Authorization code flow)* koji odgovora toku s odobrenjem autorizacijskog koda iz OAuth 2.0 specifikacije i tok predautoriziranog koda *(eng. Pre Authorized code flow)* koji nadogradnja specificirana OID4VCI specifikacijom.

4.6.1. Tok autorizacijskog koda

Tok autorizacijskog koda je isti onaj tok iz OAuth 2.0 specifikacije i koristi tip odobrenja *(eng. Grant type) authorization_code* za izdavanje pristupnih tokena koji će biti korišteni za pristup zaštićenom resursu koji je u ovom slučaju vjerodajnica.

Koraci autorizacijskog toka vidljivi na skici 4.10.:

1. Novčanik započinje tok na način da krajnji korisnik zatražuje vjerodajnicu od izdavača preko novčanika primjerice iz liste ponuđenih vjerodajnica (1a na skici 4.10.) ili izdavač započinje tok na način da generira **ponudu vjerodajnice** te je prenese na novčanik u obliku QR koda ili URI-ja (1b na skici 4.10.)

2. Novčanik dohvaća metapodatke izdavača kako bi znao lokacije krajnjih točaka, tipove vjerodajnica i podržane formate

3. Novčanik šalje autorizacijski zahtjev na autorizacijsku krajnju točku dodatno na toj krajnjoj točki se autentificira korisnik

4. Autorizacijska krajnja točka vraća odgovor s autorizacijskim kodom

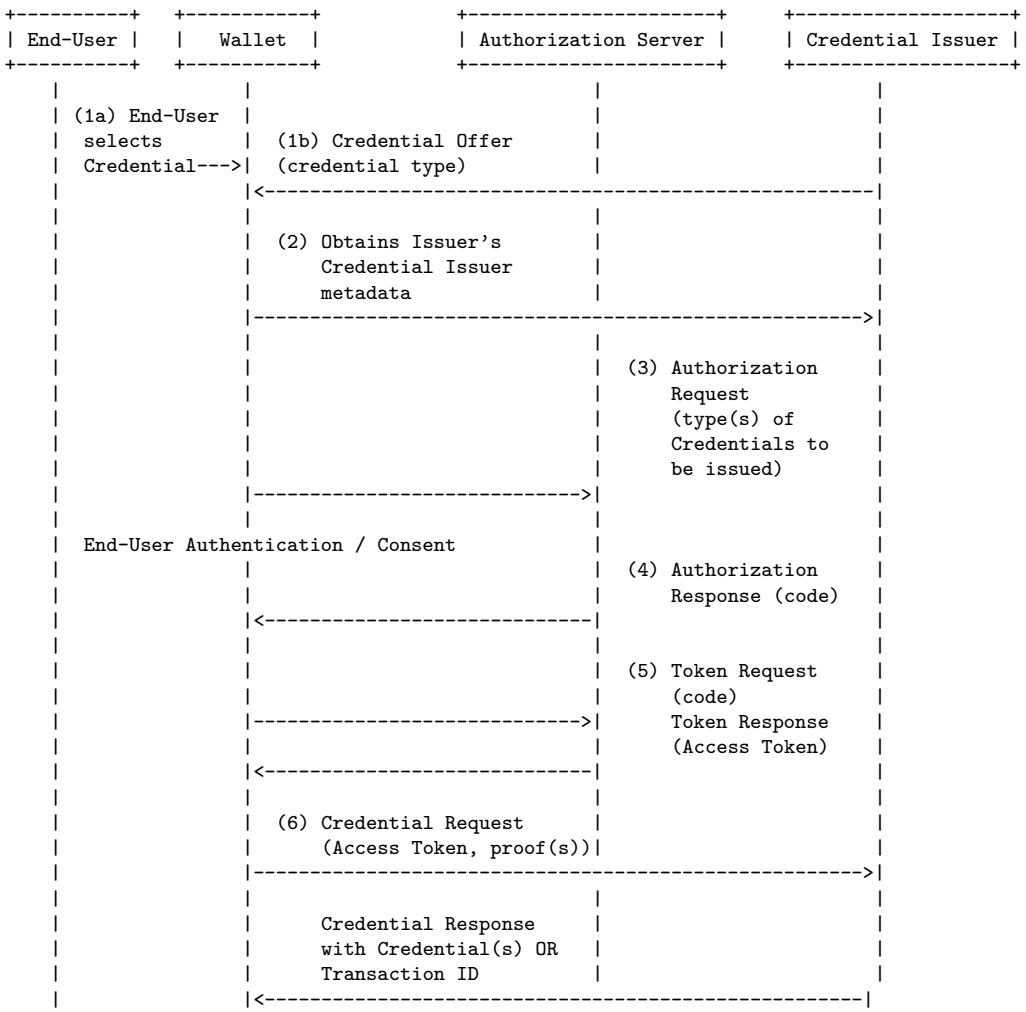
5. Novčanik šalje zahtjev za token koji sadržava autorizacijski kod na krajnju točku za izdavanje tokena te u odgovoru dobiva pristupni token

6. Konačno novčanik šalje zahtjev za vjerodajnicom koji sadržava pristupni token i dokaz o posjedu kriptografskog materijala za koji će se vezati izdana vjerodajnica te u odgovoru novčanik prima izdanu vjerodajnicu ili transakcijski ID

Ako je primljen transakcijski ID novčanik nakon nekog vremena treba razmijeniti transakcijski ID za vjerodajnicu na krajnjoj točki za odgođeno izdavanje vjerodajnica. Moguće je zatražiti više vjerodajnica od jednom koristeći krajnju točku za izdavanje skupa vjerodajnica [22].

4.6.2. Tok predautoriziranog koda

Tok predautoriziranog koda je dodatak koji specificira OID4VCI specifikacija i koristi novi tip odobrenja *(eng. Grant type)* s vrijednosti *urn:ietf:params:oauth:grant-type:pre-authorized_code*. Prije iniciranja ovog toka izdavač treba izvršiti pripremu za izdavanje



Slika 4.10. Tok autorizacijskog koda protokola OID4VCI preuzet s [22]

koja se sastoji od autentifikacije i autorizacije. Zbog toka nema potrebe za korištenjem autorizacijske krajnje točke jer je odgovornost autentifikacije i autorizacije na izdavaču. Primjerice izdavač će autentificirati budućeg posjednika vjerodajnice metodama iz fizičkog svijeta poput pregleda osobne iskaznice. Budući posjednik koje je autentificiran će primiti predautorizirani kod od izdavača na svoj novčanik. Zatim će novčanik taj predautorizirani kod direktno zamijeniti za pristupni token.

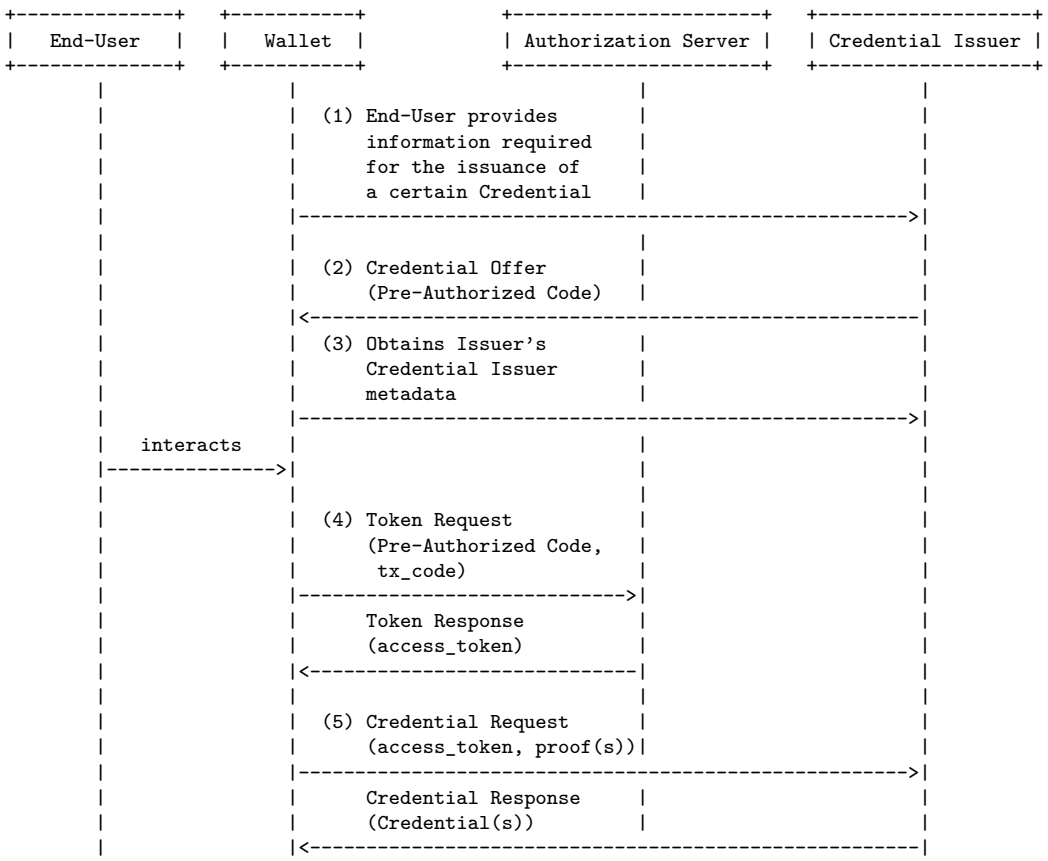
Koraci toka predautoriziranog koda vidljivi su na skici 4.11.:

1. Krajnji korisnik dostavlja informacije potrebne za izdavanje vjerodajnice izdavaču
2. Novčanik od izdavača prima ponudu vjerodajnice koja sadržava predautorizirani kod putem QR koda ili URI-ja
3. Novčanik dohvaća metapodatke izdavača
4. Novčanik šalje zahtjev za token na krajnju točku za izdavanje tokena. Zahtjev sadržava predautorizirani kod i dodatno **transakcijski kod** *(tx_code)*. U odgovoru novčanik dobiva pristupni token. Transakcijski kod je slučajno generiran broj koji se šalje krajnjem korisniku prilikom izdavanja ponude vjerodajnice. Ovaj kod se šalje drugim kanalom primjerice SMS porukom ili epoštom kako bi funkcionirao kao prevencija napada ponovnim slanjem *(eng. replay attack)* na primjer ako napadač stoji iza legitimnog krajnjeg korisnika i skenira QR kod na kojemu se nalazi ponuda i predautorizirani kod.
5. Šalje se zahtjev za vjerodajnicom isto kao i u toku autorizacijskog koda

4.6.3. Krajnja točka za izdavanje ponude vjerodajnice

Krajnja točka za izdavanje ponude vjerodajnice se koristi kako bi se krajnjem korisniku koji želi inicirati tok proslijedile informacije o ponudi vjerodajnice. Izdavač šalje ponudu koristeći HTTP GET zahtjev ili HTTP redirect na **krajnju točku novčanika za primanje ponude vjerodajnice**. Ponuda vjerodajnice sadrži jedan od sljedeća dva parametra enkodirani u URI-ju:

- **credential_offer** parametar sadržava JSON objekt s parametrima ponude



Slika 4.11. Tok predautoriziranog koda protokola OID4VCI preuzet s [22]

- **credential_offer_uri** parametar sadržava URI string koji referencira JSON objekt s parametrima ponude

Izdavač može generirati QR kod koji sadržava ponudu ili URI string na koji će krajnji korisnik kliknuti [22].

Parametri ponude u JSON objektu su [22]:

- **credential_issuer** je URL izdavača vjerodajnice
- **credential_configuration_ids** je polje stringova koji jedinstveno identificiraju vrstu neke od podržanih vjerodajnica
- **grant** je opcionalni parametar koji specificira tip odobrenja, svaki tip odobrenja je prikazan kao par ključ/stavka gdje je ključ tip odobrenja, a vrijednost su dalji parametri tog odobrenja

Ako je ključ *grant* postavljen na *authorization_code* koristit će se tok autorizacijskog koda. Pod stavku u ovom slučaju je potrebno postaviti objekt s parametrima

issuer_state i **authorization_server**. Oba parametra su opcionalna te se u prvi pohranjuje string vrijednost koja služi za uspostavljanje stanja u daljnjim zahtjevima dok se u drugi pohranjuje string koji služi za identifikaciju autorizacijskog servera ako ih je specificirano više u metapodacima izdavača [22].

Ako je ključ *grant* pak postavljen na *urn:ietf:params:oauth:grant-type:pre-authorized_code* koristit će se tok s predautoriziranim kodom. U stavku se stavlja objekt s obaveznim parametrom **pre-authorized_code** koji sadržava predautorizirani kod koji je generirao izdavač. Opcionalni parametar je i **authorization_server** koji je već prethodno opisan. Također još jedan opcionalni parametar je i **tx_code** koji specificira zahtjeve za unos transakcijskog koda poput duljine, vrsta znakova unosa(numerički ili tekstualni) te opis koji će navoditi korisnika [22].

4.6.4. Autorizacijska krajnja točka

Autorizacijska krajnja točka se koristi na isti način kao i u OAuth 2.0 uz preporuku korištenja Proof Key for Code Exchange (PKCE) mehanizma iz [23] čija je uloga prevencija napada presretanja autorizacijskog koda i preporuku korištenja POST metode direktno između klijenta i izdavača, a ne korištenjem HTTP Redirect kako bi se osigurali povjerljivost, integritet i autentičnost zahtjeva te izbjegli problemi s velikim veličinama [22].

Postoje dvije metode za zahtijevanje izdavanje vjerodajnice u autorizacijskom zahtjevu. Prva metoda je korištenje parametra *authorization_details* onako kako ga specificira [24]. U parametru će biti pohranjeni detalji o tipu vjerodajnice, formu i slično. A druga metoda koristi postojeći *scope* parametar.

4.6.5. Krajnja točka za izdavanje tokena

Krajnja točka za izdavanje tokena koristi se na isti način kao što je definirana u OAuth 2.0.

U zahtjev je potrebno uključiti dva dodatna parametra ako se radi o toku predautoriziranog koda:

- **pre-authorized_code** je kod dobiven u ponudi vjerodajnice

- **tx_code** je opcionalni parametar koji treba biti postavljen na transakcijski kod dobiven drugim kanalom ako je u ponudi specificirano da se koristi transakcijski kod

Ako se koristi tok predautorizacijski koda autentifikacija klijenta je opcionalna. Također preporučeno je da izdani pristupni token vrijedi samo za vjerodajnice iz ponude.

U odgovoru uz specificirane parametre u OAuth 2.0 specifikaciji autorizacijski server može vratiti i dodatne parametre:

- **c_nounce** je slučajna vrijednost koja će biti korištena za kreiranje dokaza o posjedu kriptografskog materijala
- **c_nounce_expires_in** je vijek trajanja *c_nounce* vrijednosti izražen u sekundama
- **authorization_details** parametar je obavezan ako je bio korišten u autorizacijskom zahtjevu

4.6.6. Krajnja točka za izdavanje vjerodajnice

Krajnja točka za izdavanje vjerodajnice izdaje vjerodajnicu nakon uspješne provjere priloženog pristupnog tokena. TLS mora biti korišten na ovoj krajnjoj točki. Izdavač bi trebao kriptografski povezati identifikator krajnjeg korisnika s vjerodajnicom koja će biti izdana što će omogućiti provjeravatelju tijekom prezentacije da se uspješno uvjeri da je krajnji korisnik uistinu taj kome je vjerodajnica namijenjena.

Zahtjev na krajnju točku treba biti poslan koristeći HTTP POST metodu i u tijelu treba sadržavati sljedeće parametre enkodirane u *application/json* formatu:

- **format** je string koji određuje format vjerodajnice koja će biti izdana te je obavezan parametar ako nije vraćen parametar *credential_identifiers* unutar *authorization_details* parametra u odgovoru s krajnje točke za izdavanje tokena
- **credential_identifier** je string koji identificira vjerodajnice koje će biti izdane te je obavezan ako je parametar *credential_identifiers* bio vraćen s krajnje točke za izdavanje tokena

- **proof** je objekt koji sadržava dokaz o posjedu kriptografskog materijala za koji će vjerodajnica biti vezana, proof objekt također mora sadržavati **proof_type** parametar koji specificira tip dokaza, nadalje vrijednost ovog parametra specificira naziv ostalih parametara u ovom objektu, *proof* parametar ne smije biti prisutan ako je korišten parametar *proofs*

- **proofs** je objekt koji sadrži jedan ili više dokaza o posjedu kriptografskog materijala te on ne smije biti prisutan ako je prisutan *proof* objekt

- **credential_response_encryption** je opcionalni parametar koji sadrži podatke o enkripciji odgovora na ovaj zahtjev

proof parametar mora sadržavati identifikator izdavača i *c_nounce* vrijednost koju je generirao autorizacijski server. Način na koji je moguće staviti ove vrijednosti u dokaz jest primjerice koristeći JWT. Identifikator izdavača će biti zapisan u *aud* tvrdnji, a *c_nounce* će biti postavljen u *nounce* tvrdnji. Te će konačno token biti potpisan privatnim ključem krajnjeg korisnika.

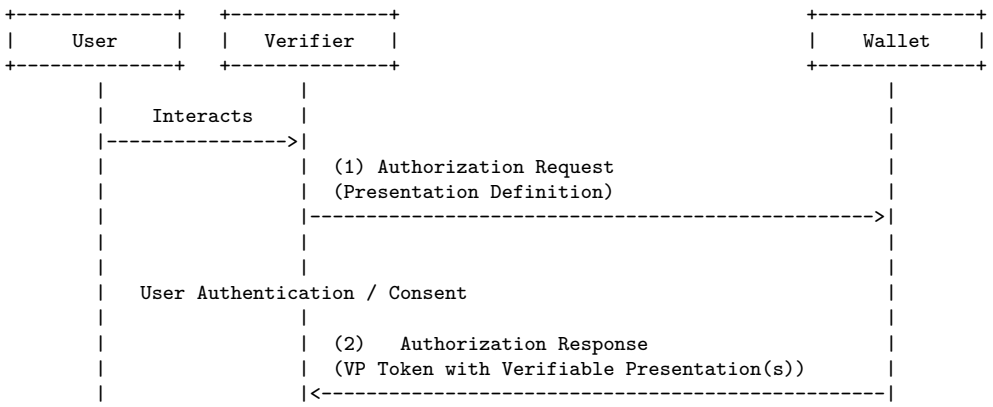
Nadalje ako je odgovor enkriptiran media tip treba biti postavljen na *application/jwt* inače na *application/json*. U odgovoru potrebno je vratiti sljedeće parametre:

- **credential** parametar treba sadržavati izdanu vjerodajnicu može biti ili string ili objekt ovisno o formatu

- **credentials** sadržava polje izdanih vjerodajnica

- **transaction_id** treba sadržavati identifikator koji će biti razmijenjen za vjerodajnicu na krajnjoj točki za odgođeno izdavanje vjerodajnice nakon nekog vremena **c_nounce** je opcionalni parametar koji je string koji je korišten za kreiranje dokaza o kriptografskom materijalu **c_nounce_expires_in** opcionalni parametar koji izražava vijek trajanja *c_nounce* vrijednosti izražen u sekundama **notification_id** je opcionalni parametar koji je string koji identificira izdanu vjerodajnicu te novčanik ovaj parametar uključuje u zahtjev za obavijesti, parametar ne smije biti prisutan ako nije prisutan parametar *credential*

Samo jedan od prva tri parametra smije biti prisutan u odgovoru.



Slika 4.12. Tok protokola OID4VP kada su oba entiteta na istom uređaju preuzet s [25]

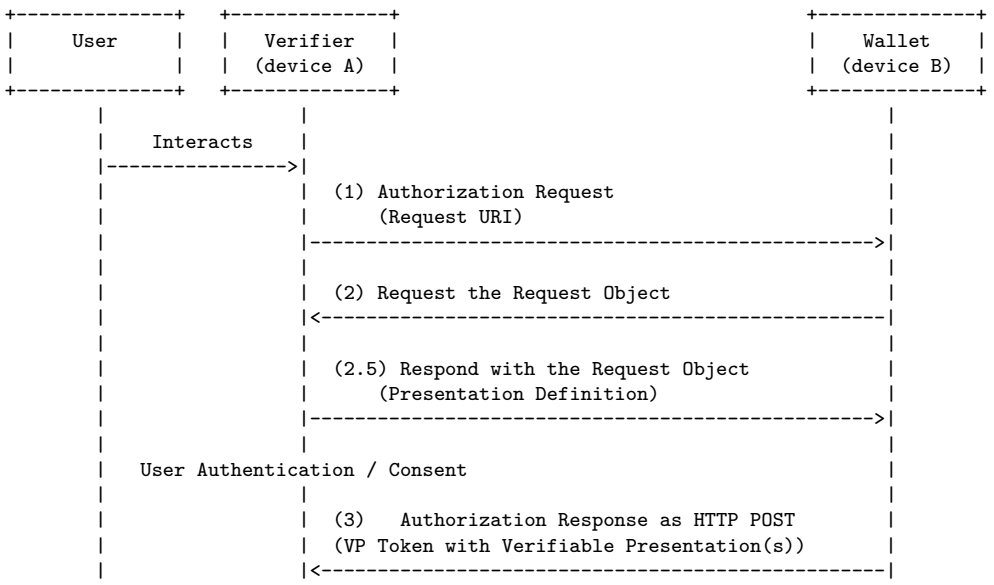
4.7. OpenID for Verifiable Presentations

OpenID for Verifiable Presentations (OID4VP) je protokol za zahtijevanje i izdavanje provjerljivih prezentacija. Specifikacija nadograđuje OAuth 2.0 protokol mehanizmom za prezentiranje provjerljivih vjerodajnica u obliku provjerljivih prezentacija. Nadogradnja može biti implementirana i na OpenID Connect protokolu. Protokol može biti kombiniran i sa SIOPv2 protokolom kako bi se ID tokeni mogli samostalno izdavati.

Primarno proširenje je uvođenje VP tokena kao kontejner koji omogućuje krajnjim korisnicima da predstave provjerljive prezentacije koristeći novčanik. VP token sadržava jednu ili više provjerljivih vjerodajnica u istom ili različitim formatima.

Specifikacija definira dva toka. Prvi tok je kada se provjeritelj i novčanik nalaze na jednom fizičkom uređaju. Provjeravatelj će poslati autorizacijski zahtjev na novčanik sa zahtjevima kakve vjerodajnice traži. Krajnji korisnik će odabrati pripadajuće vjerodajnice te će novčanik formirati provjerljivu prezentaciju i poslati provjeravatelju unutar URI-ja koristeći HTTP Redirect. Primjer toka je vidljiv na 4.12.

Drugi tok se koristi u slučaju kada su novčanik i provjeravatelj na dva različita fizička uređaja. U tom slučaju provjeravatelj pripremi autorizacijski zahtjev i prikaže ga kao QR kod koji će korisnik skenirati koristeći novčanik. Prezentacija će biti pripremljena i poslana na URL koji kontrolira provjeravatelj koristeći HTTP POST metodu. Ako je autorizacijski zahtjev prevelik da bi se generirao kao QR kod onda će se generirati QR kod s poveznicom na URI na kojem će biti postavljen autorizacijski zahtjev



Slika 4.13. Tok protokola OID4VP kada su oba entiteta na različitim preuzet s [25]

te će ga novčanik preuzeti koristeći HTTP GET metodu. Primjer ovakvog toka vidljiv je na 4.13.

Promatrano kroz prizmu OpenID Connect protokola na koji je moguće implementirati ovu nadogradnju, novčanik ima ulogu OpenID pružatelja identiteta (OP), a provjeravatelj ima ulogu oslanjajuće strane (RP).

4.7.1. Autorizacijski zahtjev

Autorizacijski zahtjev slijedi definiciju danu u OAuth 2.0 specifikaciji. Specifikacija definira dodatno nove parametre zahtjeva:

- **presentation_definition** je string koji sadrži definiciju prezentacije u JSON obliku, ovaj parametar mora biti prisutan ako nisu prisutni *presentation_definition_uri* parametar ili *scope* parametar koji predstavlja definiciju prezentacije
- **presentation_definition_uri** je string koji sadrži HTTPS URL koji referencira objekt definicije prezentacije u JSON obliku, ovaj parametar mora biti prisutan ako nisu prisutni *presentation_definition* parametar ili *scope* parametar koji predstavlja definiciju prezentacije
- **client_id_scheme** je opcionalni parametar koji je string i identificira shemu *client_id* parametra, ako nije prisutan novčanik se treba ponašati sukladno OAuth

2.0 specifikaciji

- **client_metadata** je JSON objekt koji sadržava metapodatke provjeritelj, ovaj parametar mora biti prisutan ako nije prisutan *client_metadata_uri* parametar
- **client_metadata_uri** je string koji sadrži HTTPS URL koji referencira JSON objekt koji sadrži metapodatke provjeritelja, ovaj parametar mora biti prisutan ako nije prisutan *client_metadata* parametar

Definicija prezentacija specificira koje tvrdnje iz koje vjerodajnice treba koristiti za kreiranje prezentacije. Moguće je predstaviti cijelu vjerodajnicu, više vjerodajnica ili pak samo pojedine tvrdnje iz jedne ili više vjerodajnica. Parametar*client_id_scheme* omogućava specificiranje različiti identifikatora izvan OAuth 2.0 specifikacije.

Sljedeća dodatna razmatranja dana su za već postojeće parametre:

- **nounce** je obavezni parametar i definiran u OpenID Core specifikaciji te bi se koristio kako bi se prezentacija generirana od novčanika povezala s određenom transakcijom odnosno kako bi se spriječio napad ponavljanja VP tokena *(eng. replay attack)*
- **scope** je opcionalni parametar i definiran u OAuth 2.0 specifikaciji te bi se koristio za zatraživanje provjerljive prezentacije
- **response_mode** je opcionalni parametar i već je definiran u OAuth 2.0 Response Type Encoding Practices specifikaciji te bi se koristio kao definiranje načina na koji će prezentacija biti poslana

Parametar *response_mode* je prethodno već definiran za dvije vrijednosti. Ako je vrijednost *query* tada će parametri će biti stavljeni u query dio URI-ja i proslijeđeni koristeći HTTP Redirect. Ako je vrijednost *fragment* također će biti korišten HTTP Redirect, ali će parametri biti enkodirani u fragment URI-ja [26]. OID4VP definira novu vrijednost *direct_post* koja omogućava da parametri budu poslani koristeći HTTP POST metodu. Nova vrijednost je uvedena kako bi se prezentacija poslala ako su provjeravatelj i novčanik različiti uređaji jer u tom slučaju nije moguće koristiti HTTP Redirect. Također ovaj način omogućava prijenos više podataka ako je URL limit

korisničkih agenata (eng. user agent) nedovoljan. Kada se koristi opcija *direct_post* obavezno je postaviti i parametar *response_uri* u autorizacijski zahtjev koji definiran URI na koji je potrebno poslati prezentaciju POST metodom. Kada je zadan ovaj parametar onda parametar *redirect_uri* ne smije biti prisutan.

Također u autorizacijskom zahtjevu postoji parametar *response_type* koji je prethodno definiran OAuth 2.0 specifikacijom. OID4VP specifikacija definira novu vrijednost tog parametra vp_token koja zahtjeva da se u odgovoru pošalje VP token, a ne primjerice autorizacijski kod.

4.7.2. Odgovor

VP token može biti vraćen u odgovoru na autorizacijski zahtjevi ako je vrijednost parametra *response_type* postavljena na *vp_token* ili *id_token* ili pak u odgovoru na zahtjev za izdavanje tokena ako je vrijednost parametra postavljena na *code*.

Kada se VP token vraća u odgovoru potrebno je postaviti dva parametra:

- **vp_token** je JSON string ili JSON objekt koji sadržava jednu ili više provjerljivih prezentacija
- **presentation_submission** je sadrži preslikavanja između provjerljivih vjerodajnica i njihove lokacije u VP tokenu

Po primanju VP tokena provjeravatelj mora uspostaviti vezu između prezentacija i vjerodajnica koristeći preslikavanje iz prezentacijskog prilaganja *(eng. Presentation Submission)*. Zatim za svaku prezentaciju provjeriti integritet, autentičnost i kriptografsku povezanost s posjednikom vjerodajnice ovisno o pravilima prezentacijskog formata. Nadalje potrebno je provjeriti jesu li ispunjeni svi kriteriji iz definicije prezentacije te napraviti dodatne provjere poput pregleda je li vjerodajnica povučena.

5. Implementacija sustava

Implementacija napravljena u sklopu ovog diplomskog rada se sastoji od dvije web aplikacije. Prva aplikacija je *Issuer* to jest izdavač provjerljivih vjerodajnica koristeći OID4VCI protokol, a druga aplikacija je *Verifier* koja je provjeravatelj provjerljivih prezentacija koristeći OID4VP protokol. Treći entitet koji nedostaje je novčanik na kojemu će se pohranjivati vjerodajnica. Kao novčanik u svrhu demonstracije se koristio open source novčanik u obliku native aplikacije za iOS platformu naziva *Sphereon Wallet* koji je dostupan za preuzimanje na [27]. Korištenje postojećeg novčanika demonstrira interoperabilnost ove dvije aplikacije napravljene u svrhu demonstracije u aplikacije napravljene od treće strane.

Obe aplikacije *Issuser* i *Verifier* su napravljene koristeći web tehnologije html, css i JavaScript. Na serverskoj strani aplikacije su napisane koristeći TypeScript koji se prevodi u JavaScript i pokreću se u okolini NodeJS. Obe aplikacije koriste biblioteku Express za postavljanje web servera i deployane su na internet koristeći servis `https://render.com/`. Kod aplikacija ne sadržava funkcionalnost TLS jer je ona automatski podešena jednom kada se aplikacija deploya na servis `https://render.com/`. Ako se ne koristi navedeni servis potrebno je osigurati TLS na svim rutama aplikacije radeći preinake u kodu ili podešavajući *reverse proxy* s TLS funkcionalnošću.

Prije uporabe sustave potrebno je generirati par kriptografskih ključeva za izdavača i provjeravatelja. Privatni ključ treba ostati tajni, dok će javni ključ biti objavljen unutar DID dokumenta. Krajnji korisnik će nakon instalacije novčanika imati vlastiti par privatnog i javnog ključa.

Kada je novčanik instaliran krajnji korisnik će pokrenuti aplikaciju *Issuer* gdje će stvoriti novu ponudu za izdavanje vjerodajnice. Nakon što je ponuda stvorena korisnik

će klikom na URI otvoriti ponudu u novčaniku ako su aplikacija *Issuer* i novčanik na istom uređaju ili će skenirati QR kod koji otvara ponudu u novčaniku ako su na različitim fizičkim uređajima. Novčanik će konzumirati ponudu i nakon određenih koraka ovisno o odabranom toku, izdavač će izdati vjerodajnicu novčaniku. Korisnik će pregledati vjerodajnicu koja je primljena i odlučiti hoće li je zadržati ili neće. Sva komunikacija se odvija koristeći OID4VCI protokol.

Od izdane provjerljive vjerodajnice moguće je generirati provjerljivu prezentaciju. U tu svrhu je izrađena aplikacija *Verifier* s kojom će novčanik komunicirati protokolom OID4VP kako bi priložio provjerljivu prezentaciju. Korisnik pokreće aplikaciju i na njoj kreira autorizacijski zahtjev koji će novčanik konzumirati preko URI-ja ili QR koda. Kada je autorizacijski zahtjev prenesen na novčanik, krajnjem korisniku se prikaže izbornik gdje odaberi vjerodajnice koje zadovoljavaju definiciju prezentacije iz autorizacijskog zahtjeva. Nakon odabira vjerodajnica generira se provjerljiva prezentacije koja se šalje provjeravatelju. Aplikacija *Verifier* će cijelo vrijeme prikazivati status autorizacijskog zahtjeva i nakon što je dobiveni odgovor provjeren ispisat će podatke koji su vidljivi iz provjerljive prezentacije.

5.1. Generiranje kriptografskih ključeva i postavljanje decentraliziranih identifikatora

Za potrebe generiranja kriptografskih ključeva izrađena je skripta E koja generira par privatnog i javnog ključa te ih sprema u datoteke u različitim formatima.

Javni ključ u formatu JSON Web Key (JWK) je potrebno kopirati i spremiti u DID dokument pod parametrom *publicKeyJwk* kao što je vidljivo na primjeru F.1 Navedeni DID dokument treba biti negdje pohranjen primjerice na Ethereum lancu blokova ili na internetu. Ako pogledamo *id* navedenog primjera vidimo da je vrijednost identifikatora `did:web:lovremitrovic.github.io:did-database:verifier` i da je korištena DID metoda web što znači da će ovaj DID referencirati dokument pohranjen na internetskoj URL adresi `https://lovremitrovic.github.io/did-database/verifier/did.json`. Kada se DID razlučuje u DID dokument provjerava se jednakost parametra *id* unutar DID dokumenta i DID-a koji se razlučio. Na ovaj način javni ključ provjeravatelja, u slučaju prethodnog DID-a, je javno dostupan svima na internetu

te će u se u budućnosti koristiti za provjeravanje digitalnih potpisa provjeravatelja. U obje aplikacije potrebno je u varijablu okruženja (eng. environment variable) PUBLIC_KEY_DID upisati vrijednost DID-a kojeg kontrolira pripadajući identitet i koji se razlučuje u dokument s javnim ključem.

Privatni ključ treba ostati tajan i zbog toga će biti pohranjen u varijablu okruženja (eng. environment variable) u procesu na serveru koji izvršava aplikaciju *Issuer* ili *Verifier*. U slučaju aplikacije *Issuer* u varijablu PRIVATE_KEY potrebno je upisati privatni ključ u formatu PKCS8 i dodatno enkodiran u string koristeći base64 kodiranje. U slučaju aplikacije *Verifier* potrebno je u varijablu PRIVATE_KEY_HEX parametar d iz privatnog ključa zapisanog u JWK formatu dodatno enkodiran koristeći base64url kodiranje.

5.2. Issuer

Cjelovit kod aplikacije *Issuer* je dostupan na githubu [28] i u privitku A Struktura direktorija u kojem se nalazi izvorni kod prikazana je u privitku B Implementacija je u trenutku pisanja deployana na `https://openid4vc-issuer.onrender.com/`. Aplikacija implementira i tok predautoriziranog kod i tok autorizacijskog koda. U nastavku će biti opisani značajni dijelovi koda aplikacija po redoslijedu kako se on izvršava kada se korisniku izdaje vjerodajnica.

5.2.1. Početna stranica

Pri pokretanju aplikacije korisniku se dostavlja početna stranica A.4 na kojoj se nalaze dvije forme.

Prva forma šalje POST zahtjev koji stvara ponudu vjerodajnice s tokom predautoriziranog koda. Izdavač klasičnom fizičko metodom poput provjere osobne iskaznice autentificira krajnjeg korisnika i upiše adresu epošte koju kontrolira krajnji korisnik te unose podatke kojima će se napuniti vjerodajnica u ovom slučaju to je proizvođač COVID cjepiva. Ponuđena su dva proizvođača *Blue Inc.* i *Red Inc.*. Nastavak ovog toka je u poglavlju 5.2.2.

Druga forma šalje POST zahtjev koji stvara ponudu vjerodajnice s tokom autoriza-

cijskog koda. U ovom toku autentifikacija krajnjeg korisnika će se napraviti naknadno na krajnjoj točki za autorizaciju i podaci kojima se puni vjerodajnica se stvaraju na osnovu autentifikacije korisnika. Primjerice postoji baza podataka u kojoj su zabilježene osobe koje su primile COVID cjepivo zajedno s podacima o cjepivu poput proizvođača. Nastavak ovog toka je u poglavlju 5.2.3.

Također na početnoj stranici je i poveznica na metapodatke izdavača. Više o metapodacima u poglavlju 5.2.4.

5.2.2. Ponude vjerodajnice u toku predautoriziranog koda

Nakon što je primljen POST zahtjev za stvaranje ponude vjerodajnice on se obrađuje u kontroloru prikazanom u A.9

Navedeni kontroler prvo provjerava valjanost poslanih podataka u tijelu zahtjeva poput valjanosti adrese epošte i podataka koji dolaze u vjerodajnicu. Ako provjera ne prolazi vraća se odgovor s greškom *400 Bad Request*. Nadalje ako je zahtjev ispravan server generira nasumičan string koji služi kao **predautorizirani kod** koji se pohrani u varijablu *code*. Također generira se i **transakcijski kod** pohranjen u varijablu *pin* koji će biti poslan drugim kanalom.

Zatim se kreira sesija koja sprema **ponudu vjerodajnice**, transakcijski kod i predautorizirani kod za uporabu u budućnosti. Sesija će biti referencirana predautorizacijskim kodom u upravitelju sesija koji je pohranjen u objektu klase *VcIssuer*.

Nakon kreiranja sesije aplikacija šalje transakcijski kod na epoštu koja je bila unesena u formu. Funkcija koja pruža funkcionalnost slanja epošte vidljiva je u A.15 Preduvjet je postojanje računa na Outlook365 servisu te postoje dvije varijable okruženja *EMAIL_USER* i *EMAIL_PASS* koje sadržavaju korisničko ime i zaporku računa.

Ako je slanje epošte uspješno odgovor na POST zahtjev je stranica generirana po predlošku A.3 te je na njoj prikazana ponuda za izdavanje vjerodajnice u obliku QR koda koji korisnik skenira novčanikom ili otvaranjem URI-ja klikom na *Open from wallet* poveznicu. Ponuda vjerodajnice sadržava tip odobrenja, predautorizacijski kod, identifikator vrste vjerodajnice koja će biti izdana i identifikator izdavača u obliku URL adrese. Nakon što novčanik konzumira ponudu on dohvaća metapodatke izdavača i šalje

zahtjev za izdavanjem tokena na server. Međutim prije slanja zahtjeva za izdavanjem tokena korisnik treba upisati transakcijski kod koji je primio epoštom u novčanik.

5.2.3. Ponude vjerodajnice u toku autorizacijskog koda

Ako je odabran tok autorizacijskog koda šalje se prazan POST zahtjev koji obrađuje kontroler A.10 Kontrolor generira stanje izdavača u obliku slučajnog stringa pohranjenom u varijabli *issuerState* koji služi za održavanje stanja s budućim zahtjevima i referenciranje stvorene ponude u upravitelju sesija.

Nakon uspješno stvorene ponude korisniku se generira stranica po predlošku jednako kao i u 5.2.2. Međutim u ovom toku ponuda sadržava stanje izdavača umjesto predautoriziranog koda.

Nakon konzumiranja ponude novčanik dohvaća metapodatke i šalje autorizacijski zahtjev na krajnju točku za autorizaciju na kojoj će biti autentificiran korisnik.

5.2.4. Dohvaćanje metapodataka

Novčanik dohvaća metapodatke s adrese koju dobije spajanjem URL-a koji je dobivan u ponudi vjerodajnice i putanje `/.well-known/openid-credential-issuer`. U metapodacima se nalaze adrese potrebnih krajnjih točaka poput krajnje točke za izdavanje vjerodajnice, identifikatora izdavača u obliku URL-a, adresa autorizacijskih servera te informacija o podržanim vjerodajnicama. o

5.2.5. Krajnja točka za autorizaciju

Kada se koristi tok autorizacijskog koda onda na krajnju točku za autorizaciju pristiže autorizacijski zahtjev. Autorizacijski zahtjev treba biti sukladan specifikaciju OAuth 2.0 [19] i specifikaciji OID4VCI [22].

Korišteni novčanik također slaže zahtjev sukladan preporuci o uporabi PKCE mehanizma iz [23]. Navedeni mehanizam postavlja dva nova parametra u autorizacijskom zahtjevu **code_challenge_method** koja određuje metodu korištenja u provjeri izazova i **code_challange** koji predstavlja kod za izazov. Svrha ovog mehanizma je zaštita od napada presretanjem autorizacijskog koda. Izdavač treba održavati vezu

između koda izazova i izdanog autorizacijskog koda.

Po dolasku autorizacijskog zahtjeva kontroler prikazan u A.7 provjerava ispravnost zahtjeva što uključuje provjeru je li izazov provodi podržanom metodom. U ovoj implementaciji jedina podržana metoda je SHA256 koju parametar **code_challenge_method** označava s "S256".

Metoda autentifikacije korisnika nije specificira niti OAuth 2.0 specifikacijom niti OID4VCI specifikacijom. Zbog jednostavnosti u ovom implementaciji se koristi autentifikacija putem HTTP Basic scheme. Također zbog jednostavnosti postoji jedan korisnik s korisničkim imenom "user" i zaporkom "user" te je su za njega preddefinirani podaci u vjerodajnici. Novčanik će nakon slanja zahtjeva preusmjeriti korisnika u preglednik gdje će biti potrebno upisati korisničko ime i zaporku kako bi se zahtjev uspješno obradio.

Nakon autentifikacije slučajno se generira autorizacijski kod koji će ubuduće referencirati sesiju te se ponuda puni s preddefiniranim podacima za punjene vjerodajnice.

Kao odgovor na autorizacijski zahtjev novčanik će dobiti Redirect s autorizacijskim kodom u query parametru.

5.2.6. Izdavanje tokena

Na krajnju točku za izdavanje tokena čiji je kod vidljiv u A.11 novčanik šalje zahtjev za izdavanjem tokena. Kako korištena biblioteka u trenutku izrade aplikacije nema implementiranu provjeru valjanosti zahtjeva za autorizacijski kod napravljena je prerada postojećih funkcija iz biblioteke kako bi se i taj slučaj obradio. Izmjene su vidljive u A.6

Zahtjev za izdavanjem tokena sadržava transakcijski kod i predautorizirani kod ako se radi o toku predautoriziranog koda. Jednakost obje vrijednosti se uspoređuje s onima pohranjenim u sesiji. Ako su jednake nastavlja se proces izdavanja tokena.

Ako se radi o autorizacijskom kodu tada zahtjev sadržava autorizacijski kod i **provjeru izazova** u parametru *code_verifier*. Obe vrijednosti se uspoređuju s vrijednostima iz pripadajuće sesije. Autorizacijski kod treba biti jednak, a za vrijednost

code_verifier treba vrijediti sljedeća relacija [23]:

```
code_challenge = BASE64URL-ENCODE(SHA256(ASCII(code_verifier)))
```

Ako su provjere uspješne izdaje se pristupni token u obliku JWT koji sadržava vrijednost autorizacijskog koda odnosno predautoriziranog koda potpisan privatnim ključem izdavača. U odgovoru se uz pristupni token šalje i nasumična vrijednost **c_nounce** koja služi novčaniku za kreiranje dokaza o posjedu kriptografskog materijala.

5.2.7. Izdavanje vjerodajnice

Krajnja točka za izdavanje vjerodajnice izdaje novčaniku provjerljivu vjerodajnicu koja u ovom kontekstu ima ulogu zaštićenog resursa. Vjerodajnica se izdaje na osnovu zahtjeva i priloženog pristupnog tokena. Navedena krajnja točka prikazana je na A.8, funkcija za provjeru tokena A.13 i funkcija za potpisivanje vjerodajnice na A.12

Prvi korak na krajnjoj točki je dohvaćanje javnog ključa izdavača iz DID dokumenta. Zatim slijedi provjera priloženog pristupnog tokena pregledom potpisa koristeći javni ključ izdavača. Ako potpis nije valjan novčaniku se šalje greška međutim ako je potpis valjan vrijednost predautoriziranog koda odnosno autorizacijskog koda će se koristiti kako bi se dohvatili podacima kojima će se puniti vjerodajnica.

Drugi korak je provjera zahtjeva za izdavanjem vjerodajnice. Neznačajni segment provjere je provjera dokaza o posjedu kriptografskog materijala. U ovom slučaju provjerava se je li primljena vrijednost *nounce* jednaka prethodno poslanoj vrijednosti *c_nounce* te je li ona potpisana privatnim ključem čiji je pripadajući javni ključ priložen u zaglavlju.

Konačno se kreira vjerodajnica naslovljena na identifikator krajnjeg korisnika koji je ujedno, u ovom slučaju, javni ključ prethodno dostavljen u dokazu. Vjerodajnica je provjerljiva jer je potpisana privatnim ključem izdavača.

Novčanik u odgovoru prima vjerodajnicu i odlučuje hoće li je zadržati.

5.3. Verifier

Cjelovit kod aplikacije *Verifier* je dostupan na githubu [29] i u privitku C Struktura direktorija izvornog koda dana je u D te je implementacija u trenutku pisanja deployana na `https://openid4vc-verifier.onrender.com/`. Po uzoru na prethodni dio značajni dijelovi koda bit će opisani po redoslijedu izvršavanja u slučaju prilaganja jedne provjerljive prezentacije.

5.3.1. Kreiranje autorizacijskog zahtjeva

Na početnoj stranici se nalazi jedna forma čiji POST zahtjev kreira autorizacijski zahtjev. Kada server zaprimi POST zahtjev njega obrađuju kontroler C.11 koji kreira autorizacijski zahtjev i postavlja ga na zaseban URL te konačno vraća generiranu stranicu na kojoj se nalazi QR kod i URI koji otvarana novčanik i dohvaća autorizacijski zahtjev s adrese u parametru *request_uri*. Razlog zbog kojega parametri autorizacijskog zahtjeva nisu proslijeđeni u generiranom URI-ju odnosno QR kodu već se dohvaćaju sa zasebnog URL je veličina autorizacijskog zahtjeva koja čini QR kod nepraktičnim.

U ovom koraku se kreira i poveznica na kojoj će biti dostupan autorizacijski zahtjev. Takva poveznica mora biti generirana s dostatnom dozom entropije i biti dostupna kratki period vremena ako nisu dodani ostali mehanizmi kontrole pristupa [30]. U ovoj implementaciji poveznica će biti formata `/auth-req/:reference` gdje je vrijednost *:reference* zamijenjena stringom s dovoljnom dozom entropije. Sukladno preporukama iz [31] ta vrijednost mora biti duga barem 128 bitova i konstruirana koristeći kriptografski nasumičan broj. Ovakav oblik poveznice naziva se **Capability URLs** i njihova uloga je objavljivanje informacije na opskurnom ili teško pogodljivom URL kako bi informacije bile pružene isključivo entitetima pravo pristupa u obliku poznavanja poveznice.

U tu svrhu za implementaciju kreirana je klasa *CapabilityUrlsManager* koja pruža mogućnost isteka poveznice nakon 5 minuta. Entropija je generirana u kontroleru koji obrađuje zahtjev za izradu autorizacijskog zahtjeva koristeći *randomBytes* metodu iz modula *node:crypto* koja generira 32 kriptografski nasumična bajta te ih pohranjuje u varijablu *reference* koja će referencirati *correlationId* koji preko funkcionalnosti biblioteke referencira autorizacijski zahtjev.

Uz varijablu *reference* još su slučajno generirane vrijednost *correlationId* koji služi za referenciranje zahtjeva na serveru, *nounce* koji služi za kreiranje dokaza o posjedu kriptografskog materijal i *state* koji služi za održavanje stanja.

5.3.2. Dohvaćanje autorizacijskog zahtjeva

Novčanik će po skeniranju QR koda odnosno konzumiranja URI-ja dohvatiti autorizacijski zahtjev s poveznice navedene u parametru *request_uri*. U ovoj implementaciji autorizacijski zahtjev biće dohvaćen u obliku JWT potpisanom od strane provjeravatelja. Autorizacijski zahtjev sadržava podatke o definiciji prezentacije koju je potrebno priložiti kao i adresu na koju se prezentacija prilaže u parametru *response_uri* te metodu kojom se prezentacija prilaže u parametru *response_mode*.

5.3.3. Odgovaranje na autorizacijski zahtjev

Novčanik kada primi autorizacijski zahtjev daje korisniku mogućnost odabira vjerodajnica koje zadovoljavaju definiciju. Od odabranih vjerodajnica se kreira prezentacija koja se šalje u odgovoru unutar VP tokena. Uz VP token poslani su i ID token iz specifikacije OpenID Connect [20] i parametar *state* iz OAuth 2.0 specifikacije [19].

5.3.4. Dohvaćanje statusa

U demonstracijske svrhe kreirana je i ruta na koju preglednik šalje **correlationId**, a server vraća podatke o statusu prezentacije i o samoj prezentaciji ako je zaprimljena. Kontroler C.8 obrađiva zahtjeve o statusu koje šalje preglednik. A na pregledniku se pokreće skripta C.4 koja očitava *correlationId* te ga šalje na server i po zaprimanju odgovora prikazuje podatke o statusu prezentacije poput valjanosti prezentacije, podataka iz vjerodajnice i valjanosti vjerodajnice.

6. Sigurnost protokola OID4VC

U ovom poglavlju prvo se daje se pregled poznatih napada na OID4VCI i OID4VP protokole u 6.1. zatim se u 6.2. izrađuje formalni model protokola te se testiraju određena sigurnosna svojstva koristeći alat Tamarin Prover.

6.1. Poznati napadi na OID4VC

U radu [32] Hauck donosi 5 napada otkrivenih njegovom formalnom analizom protokola OID4VCI i OID4VP. Otkriveni napadi nisu iznenađujući onima koji su upućeni u specifikacije protokola jer su neki već spomenuti u samim specifikacija. Otkriveni su napadi koji narušavaju svojstvo autentičnosti izdavanje i prezentaciju vjerodajnice te integriteta sesije izdavanja i prezentacije.

Svojstvo autentičnosti u kontekstu prezentacije vjerodajnice neformalno znači da se napadač ne može predstaviti kao iskreni korisnik iskrenom provjeravatelju. Odnosno u kontekstu izdavanja vjerodajnice neformalno znači da napadač ne može koristiti vjerodajnicu iskrenog korisnika. Vjerodajnicu napadač može koristiti ukoliko kontrolira kriptografski materijal za koji je vezana vjerodajnica [32].

Svojstvo integriteta sesije u kontekstu prezentacije vjerodajnice neformalno znači da je korisnik eksplicitno zatražio prijavu kod iskrenog provjeravatelja i da se uspješno prijavio s identitetom koji je izabrao u novčaniku. U kontekstu izdavanja vjerodajnice svojstvo integriteta sesije znači da je korisnik eksplicitno zatražio izdavanje vjerodajnice i da je identitet u vjerodajnici uistinu identitet kojim se korisnik autentificirao kod izdavača [32].

6.1.1. Napad na svojstvo autentičnosti u toku predautoriziranog koda protokola OID4VCI

U toku predautoriziranog koda korisnik treba skenirati QR kod ili otvoriti URI koji u sebi sadržava ponudu za izdavanje u kojoj je predautorizirani kod. URI sadrži prilagođenu schemu *openid-credential-offer://* koja otvara aplikaciju novčanika. Međutim ne postoje restricije na operativnom sistemu koje se aplikacije mogu registrirati za otvaranje ove prilagođene scheme. Stoga je moguće da korisnik umjesto iskrenog novčanika odabere malicioznu aplikaciju koja će konzumirati ponudu i saznati predautorizirani kod.

OID4VCI specifikacija uvodi i transakcijki kod odnosno PIN koji se dostavlja drugim kanalom. Međutim PIN štiti korisnika samo ukoliko napadač primjerice skenira QR kod prije iskrenog korisnika. Međutim ukoliko korisnik konzumira ponudu koristeći zlonamjernu aplikaciju ona može tražiti korisnika da upiše PIN te na taj način saznati i PIN.

U trenutku kada napadač zna predautorizirani kod i PIN može ih zamijeniti zamijeniti za pristupni token i *c_nounce*. Zatim napadač prilaže pristupni token i dobija vjerodajnicu koja je vezana za napadačev kriptografski materijal [32].

6.1.2. Napad na svojstvo autentičnosti u toku autorizacijskog koda protokola OID4VCI

Sljedeći napad također podrazumijeva instaliranu zlonamjernu aplikaciju na korisničkom uređaju primjerice igricu u koju će se korisnik prijaviti vjerodajnicom prije igranja međutim umjesto preusmjeravanja na novčanik, korisnik će biti preusmjeren na stranicu izdavača. Napad računa na činjenicu da korisnik ne obraća pozornost i da se prijavljuje kod izdavača koji ga presumjerava natrag u aplikaciju akupa s autorizacijskim zahtjevom. S obzirom da zlonamjerna aplikacija u ovom trenutku zna parametre *client_ID*, *redirect_URI*, *code_verifier* i *code* može ih zamijeniti za pristupni token s kojim će napadač dohvatiti vjerodajnicu i vezati je za svoj kriptografski materijal [32].

Uporaba zaštititnog mehanizma PKCE [23] ne onemogućava napad jer je autorizacijski zahjev iniciran od strane maliciozne aplikacije.

6.1.3. Napad na svojstvo integriteta sesije u toku predautoriziranog koda protokola OID4VCI

U ovom napadu zlonamjerni akter koristi ponudu za izdavanje vjerodajnice u toku predautoriziranog koda kako bi napunio iskrenom korisniku izdao vjerodajnicu povezanu s kriptografskim materijalom iskrenog korisnika, ali podacima koje odabrire napadač.

Napad se izvodi tako da napadač inicira tok kod izdavača od kojeg dobije ponudu vjerodajnice. Zatim ponudu vjerodajnice u obliku primjerice QR koda dostavi iskrenom korisniku koristeći *phising* napad. Korisnik dalje nastavlja tok i vezuje izdanu vjerodajnicu za svoj kriptografski materijal [32].

6.1.4. Napad na svojstvo autentičnosti u toku s različitim uređajima u protokolu OID4VP

U toku s različitim uređaijima provjeravatelj i novčanik se nalaze na različitim fizičkim uređajim što otvara problem prebacivanja sesije s jednog uređaja na drugi. Napad je izvediv tako da napadač inicira autentifikaciju na provjeritelju. Provjeritelj mu šalje autorizacijski zahtjev koji napadač tehnikom *phising* uvaljuje iskrenom korisniku koji će neznajući poslati provjeritelju odgovor s svojom prezentacijom koja ga autentificira. Provjeritelj je sada zahtjev koji je pokrenuo napadač prepoznao kao autentifikaciju iskrenog korisnika. U prevodu sada se napadač autentificirao kao iskreni korisnik na provjeritelju. Ovaj napad je poznat u specifikaciji OID4VP protokola i protiv njega nema praktičnih popravaka [32].

6.1.5. Napad na svojstvo integriteta sesije u protokolu OID4VP

Slično kao i u 6.1.1. generirani URI koji sadržava autorizacijski zahtjev koristi prilagođenu shemu *openid4vp://* te se bilo koja aplikcija može registrirati da konzumira takav URI. Ako URI konzumira zlonamjerna aplikacija onda ona može poslati odgovor na autorizacijski zahtjev sa vjerodajnicam koje on posjeduje. U prevodu na provjeravatelj je primio vjerodajnicu od napdača misleći da se radi o iskrenom korisniku [32].

6.2. Formalni dokaz sigurnosti

Po uzoru na prethodne napade opisane u 6.1. u svrhu ovog rada pokušat će se naći napadi ili utvrdi sigurnosna svojstva protokola pomoću alata Tamarin Prover opisanim u 2.4.1. U nastavku su opisani model protokola, sigurnosna svojstva i rezultati formalne analize. Kod u Tamrinu je dostupan u G.1

6.2.1. Model

U modelu radi jednostavnosti prvo uvodimo ograničenje *OneKeyPairPerEntity* koji ograničava svaki entitet poput izdavača da posjeduje samo jedan par kriptografskih ključeva. Prva tri pravila *Register_pk*, *Get_pk* i *Reveal_ltk* modeliraju infrastrukturu javnog ključa na način da kada entitet zatraži registraciju javnog ključa u multiskup se postavljaju njegov privatni odnosno javni ključ i dovode ga u vezu s identifikatorom identiteta.

Sljedeća dva pravila *ChanOut_S* i *ChanIn_S* modeliraju sigurni kanal koji je u praksi primjerice ako se radi o prenesu podataka na internetu upravo TLS. Sigurni kanal omogućuje da je poruka dođe do primatelja neizmijenjena i da je dobije samo odabrani primatelj. Sigurni kanal je pretpostavljen u svim koracima izmjene poruka.

U nastavku se modelira OID4VCI protokol koristeći isključivo tok predautoriziranog koda. Pravilo *User_create_offer* modelira POST zahtjev na server izdavača koji stvara ponudu vjerodajnice s nekim tvrdnjama i prikazuje korisniku ponudu vjerodajnice primjerice u QR kodu.*Issuer_credential_offer* modelira preuzimanje ponude koja sadrži predautorizirani kod na novčanik primjerice skeniranjem QR koda. . *Wallet_token_req* modelira slanja zahtjeva za izdavanjem tokena tako da novčanik šalje predautorizirani kod izdavaču. *Issuer_token_res* modelira izdavanje pristupnog tokena novčaniku na način da pošalje token koji je zapravo potpisani predautorizirani kod koristeći privatni ključ izdavača skupa sa nounce vrijednosti koja je nasumično generirana.*Wallet_credential_request* pravilo modelira slanje zahtjeva za izdavanjem vjerodajnice od novčanika prema izdavaču na način da novčanik potpiše primljenu nounce vrijednost stvarajući dokaz o posjedu kriptografskog materijala te ga pošalje skupa sa tokenom. *Issuer_credential_response* modelira izdavanje vjerodajnice ako je token uistinu valjani potpis koda i ako je dokaz uistinu valjani potpis nounce

vrijednosti. Vjerodajnica je u modelu potpisana od strane izdavača, a njezin sadržaj su tvrdnje, javni ključ izdavača i javni ključ novčanika koji služe kao identifikatori. *Wallet_accept_credential* modelira prikazivanje vjerodajnice korisniku te konačno *User_accept_credential* modelira prihvaćanje vjerodajnice i spremanje u novčanik.

Sljedeća pravila modeliraju OID4VP specifikaciju na način na koji je ona implementirana u implementaciji ovoga rada. *User_Create_Presentation* pravilo modelira zahtjev za stvaranjem autorizacijskog zahtjeva. *Verifier_shows_path_to_auth_req* modelira skeniranje prikazanog QR koda koji sadržava poveznicu na autorizacijski zahtjev. *Wallet_asks_for_auth_req* modelira slanje GET zahtjev na adresu na kojoj se nazali autorizacijski zahtjev. *Verifier_sends_auth_req* modelira odgovor na prethodni zahtjev. Odgovor sadržava autorizacijski zahjtev koji u sebi sadrži potpisane parametre state i nounce. *Wallet_sends_auth_res* sadrži slanje odgovora na primljeni autorizacijski zahtjev. Odgovor sadržava potpisane vrijednosti state, nounce, javni ključ novčanika te vjerodajnicu. Konačno *Verifier_consumes_response* provjerava je li odgovor na autorizacijski zahtjev potpisan od strane novčanika i je li vjerodajnica potpisana od strane izdavača.

6.2.2. Sigurnosna svojstva

Definirane su tri leme od kojih prva služi za provjeru izvođenja toka, a preostale dvije definiraju sigurnosna svojstva.

Prva lema definirana u modelu je *executable* koja služi za provjeru je li postoji trag koji izvede protokol do kraja.

Druga lema je *autentific_claims* koja provjerava svojstvo integriteta sesije tijekom izvođenja protokola OID4VCI. Lema kaže da za svaku izdanu vjerodajnicu s određenim tvrdnjama treba postojati iniciranje toka za izdavanje vjerodajnice s tim istim tvrdnjama u trenutku koji prethodi izdavanju vjerodajnice.

Treća lema *secret* zahtjeva da su parametri code, token i c_nounce u kontekstu OID4VCI protokola tajni te da su parametri corrId, state, nounce i reference u kontekstu OID4VP protokola također tajni. Tajnost ovih parametara osigurava sigurnosno svojstvo autentičnosti protokola OID4VCI odnosno OID4VP.

6.2.3. Rezultati

Rezultat formalna analize moguće je reproducirati koristeći kod iz G.1 Kada je ovaj kod dan na ulaz Tamarina on je uspješno konstruirao dokaze za sve navedene leme i nije pronašao niti jedan napad. Razlog za nemogućnost pronalaska napada je jaka pretpostavka sigurnoga kanala koja nije uvijek provedena u stvarnosti i činjenica da model ne uključuje primjerice instalaciju zlonamjerne aplikacije na korisničkom uređaju.

7. Zaključak

EUDI okvir pruža veliki iskorak prema samostalnom suverenom identitetu koji konačno omogućava korisniku potpunu kontrolu i vlasništvo nad svojim identitetom koristeći nove protokole i specifikacije poput decentraliziranih identifikatora, OID4VCI, OID4VP i drugih. EUDI dodatno olakšava korištenje identiteta izdanog od državnih organa u privatnom sektoru pružajući jednostavan i unificiran način korištenja identiteta na internetu.

Implementacija pružena u sklopu ovog rada demonstrira način korištenja nekih od protokola koje zahtjeva EUDI okvir. Dodatno se demonstrira kompleksnost jednog ovakvog sustava koji korisniku omogućava jednostavno korištenje. Iako bit će potrebno izvjesno vrijeme dok se korisnici nauče koristiti ovakav sustav identifikacije.

Glavni problem sigurnosti dva temeljna protokola koje specificira EUDI (OID4VCI i OID4VP) leži u nepažnji korisnika. S obzirom na to da taj segment izlazi iz tehničke specifikacije protokola i prelazi više u društveno pitanje, navedeni protokoli se mogu smatrati uspješnim. Gledano iz tehničke strane glavni segment koji omogućava prevaru korisnika je prelazak sesije s jednog entiteta na drugi. U svrhu osnaživanja sigurnosti potrebno je educirati krajnje korisnike i dodatno napraviti preinake na razini preglednika i operacijskog sustava, ali to izlazi iz okvira navedenih specifikacija.

Literatura

[1] "APA Dictionary of Psychology — dictionary.apa.org", https://dictionary.apa.org/identity, [Accessed 24-05-2024].

[2] K. Cameron, "The laws of identity", Nov 2005. [Mrežno]. Adresa: https://www.identityblog.com/stories/2005/05/13/TheLawsOfIdentity.pdf

[3] C. Allen, "The path to self-sovereign identity", Apr 2016. [Mrežno]. Adresa: https://www.lifewithalacrity.com/article/the-path-to-self-soverereign-identity/

[4] "Definition of CREDENTIAL — merriam-webster.com", https://www.merriam-webster.com/dictionary/credential, [Accessed 24-05-2024].

[5] Microsoft Corporation i Microsoft Press, *Microsoft Computer Dictionary*, 5. izd. Redmond, WA: Microsoft Press, svibanj 2002.

[6] "What is Authentication? Definition of Authentication, Authentication Meaning - The Economic Times — economictimes.indiatimes.com", https://economictimes.indiatimes.com/definition/authentication, [Accessed 27-05-2024].

[7] "What is ICANN, and How is it Related to Registries and Registrars? - DomainTools | Start Here. Know Now. — domaintools.com", https://www.domaintools.com/support/what-is-icann-and-how-is-it-related-to-registries-and-registrars/, [Accessed 03-06-2024].

[8] E. Team, "Digital Certificate: A Comprehensive Guide - NETWORK ENCYCLOPEDIA — networkencyclopedia.com", https://networkencyclopedia.com/digital-certificate/, [Accessed 06-06-2024].

[9] “Formal Verification - an overview | ScienceDirect Topics — sciencedirect.com”, https://www.sciencedirect.com/topics/computer-science/formal-verification, [Accessed 23-06-2024].

[10] “Tamarin-prover manual”, https://tamarin-prover.com/manual/master/tex/tamarin-manual.pdf, 2024., [Accessed 23-06-2024].

[11] X. Hofmeier, “Formal analysis of web single-sign on protocols using tamarin”, Bachelor thesis, Swiss Federal Institute of Technology Zurich, 2019. [Mrežno]. Adresa: https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/ba-19-hofmeier-oidc.pdf

[12] E. Christopher, “How to wisely identify natural subjects in verifiable credentials”, Bachelor thesis, Technical University of Munich, 2023. [Mrežno]. Adresa: https://wwwmatthes.in.tum.de/file/68d8tf0myhqj/Sebis-Public-Website/Student-Theses-Guided-Research/Current-Theses-Guided-Researches/Bachelor-s-Thesis-Evan-Christopher/231015%20BA%20Thesis%20Evan%20Christopher.pdf

[13] “eIDAS Regulation — digital-strategy.ec.europa.eu”, https://digital-strategy.ec.europa.eu/en/policies/eidas-regulation, [Accessed 06-06-2024].

[14] “European Digital Identity (EUDI) Regulation — digital-strategy.ec.europa.eu”, https://digital-strategy.ec.europa.eu/en/policies/eudi-regulation, [Accessed 06-06-2024].

[15] “The european digital identity wallet architecture and reference framework”, https://ec.europa.eu/newsroom/dae/redirection/document/93678, 2023.

[16] D. Reed, M. Sabadello, A. Guy, i M. Sporny, “Decentralized identifiers (DIDs) v1.0”, W3C, W3C Recommendation, srpanj 2022., https://www.w3.org/TR/2022/REC-did-core-20220719/.

[17] “DID Methods | Performant and modular APIs for Verifiable Data and SSI — veramo.io”, https://veramo.io/docs/veramo_agent/did_methods/#:~:text=did%3Aethr%20%E2%80%8B&text=Ethr%2DDID%20provides%20a%20scalable,smart%20contract%20based%20identity%20methods., [Accessed 13-06-2024].

[18] T. T. Jr, G. Cohen, M. Sporny, I. Herman, i M. Jones, "Verifiable credentials data model v2.0", W3C, Candidate Recommendation, svibanj 2024., https://www.w3.org/TR/2024/CRD-vc-data-model-2.0-20240513/.

[19] D. Hardt, "The OAuth 2.0 Authorization Framework", RFC 6749, listopad 2012. https://doi.org/10.17487/RFC6749

[20] "Final: OpenID Connect Core 1.0 — openid.net", https://openid.net/specs/openid-connect-core-1_0-final.html, 2014.

[21] "Openid for verifiable credentials", https://openid.net/wordpress-content/uploads/2022/06/OIDF-Whitepaper_OpenID-for-Verifiable-Credentials-V2_2022-06-23.pdf, [Accessed 19-06-2024].

[22] T. Lodderstedt, K. Yasuda, i T. Looker, "Openid for verifiable credential issuance", https://openid.net/wordpress-content/uploads/2022/06/OIDF-Whitepaper_OpenID-for-Verifiable-Credentials-V2_2022-06-23.pdf, jun 2024., [Accessed 19-06-2024].

[23] N. Sakimura, J. Bradley, i N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, rujan 2015. https://doi.org/10.17487/RFC7636

[24] T. Lodderstedt, J. Richer, i B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, svibanj 2023. https://doi.org/10.17487/RFC9396

[25] O. Terbu, T. Lodderstedt, K. Yasuda, i T. Looker, "OpenID for Verifiable Presentations - Editor's draft — openid.github.io", https://openid.github.io/OpenID4VP/openid-4-verifiable-presentations-wg-draft.html, jun 2024., [Accessed 19-06-2024].

[26] E. B. de Medeiros, M. Scurtescu, P. Tarjan, i M. Jones, "Final: OAuth 2.0 Multiple Response Type Encoding Practices — openid.net", https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html, [Accessed 21-06-2024].

[27] "Sphereon Wallet", https://apps.apple.com/us/app/sphereon-wallet/id1661096796.

[28] L. Mitrović, "Implementation of issuer using OID4VC protocol", lipanj 2024. [Mrežno]. Adresa: https://github.com/LovreMitrovic/openid4vc-issuer

[29] ——, "Implementation of verifier using OID4VC protocol", lipanj 2024. [Mrežno]. Adresa: https://github.com/LovreMitrovic/openid4vc-verifier

[30] N. Sakimura, J. Bradley, i M. B. Jones, "The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)", RFC 9101, kolovoz 2021. https://doi.org/10.17487/RFC9101

[31] T. Lodderstedt, M. McGloin, i P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, siječanj 2013. https://doi.org/10.17487/RFC6819

[32] F. Hauck, "Openid for verifiable credentials: Formal security analysis using the web infrastructure model", diplomski ili magistarski rad, University of Stuttgart, 2023. [Mrežno]. Adresa: https://elib.uni-stuttgart.de/bitstream/11682/13791/1/Masterarbeit_Hauck_Fabian.pdf

Sažetak

Protokoli za izdvanje i prezentaciju vjerodajnica u EUDI okviru

univ. bacc. ing. comp. Lovre Mitrović

U radu se objašnjava pojam identiteta na internetu i njegov razvoj kroz povijest interneta. Zatim slijedi objašnjenje regulacije eIDAS i EUDI okvira. Nakon uvida u specifikacije donosi se pregled protokola koje zahtjeva EUDI specifikacija s naglasnom na OID4VC skup protokola. Kako bi oni razumjeli dan je pregled i u protokole OAuth 2.0 i OIDC na koje se nadograđuje ovaj skup protokola. Razvijena je implementacija koja koristi OID4VCI i OID4VP protokol kako bi se demonstriralo njihovo korištenje. Konačno razmotrena su sigurnosna svojstva navedena dva protokola i napravljena je njihova nalaza koristeći alat Tamarin.

Ključne riječi: identitet, samostalni suvereni identitet, eIDAS, EUDI, DID, OAuth 2.0, OpenId Connect, OpenId for Verifiable Credentials, Tamarin, formalna analiza sigurnosnih svojstava

Abstract

Protocols for issuing and presenting credentials in the EUDI framework

univ. bacc. ing. comp. Lovre Mitrović

The paper explains the concept of identity on the internet and its development through the history of the internet. It then follows with an explanation of the eIDAS regulation and the EUDI framework. After providing an overview of the specifications, the paper presents a review of the protocols required by the EUDI specification, with an emphasis on the OID4VC set of protocols. To aid understanding, a review of the OAuth 2.0 and OIDC protocols, on which this set of protocols builds, is provided. An implementation was developed using the OID4VCI and OID4VP protocols to demonstrate their use. Finally, the security properties of these two protocols were considered and their analysis was conducted using the Tamarin tool.

Keywords: identity, self-sovereign identityeIDAS, EUDI, DID, OAuth 2.0, OpenId Connect, OpenId for Verifiable Credentials, Tamarin, formal analysis of security properties

Privitak A: Kod Issuer aplikacije

Listing A.1: Sadržaj datoteke ./issuer/src/app.ts

```
import dotenv from 'dotenv';
import express from 'express';
import indexRouter from './router';
import path from "node:path";
import {initIssuer} from "./service/issuer";
dotenv.config();

const externalUrl = process.env.RENDER_EXTERNAL_URL;
const port = process.env.PORT ? parseInt(process.env.PORT) : 3000;

const app = express();

app.use(express.urlencoded({ extended: true })); // support encoded
    bodies
app.use(express.json());
app.use(express.static(path.join(__dirname, 'public')));
app.set('view engine','ejs');
app.set('views', path.join(__dirname, 'views'))

app.use('/', indexRouter);

app.locals.symmetricKey = process.env.SYMMETRIC_KEY;

if(externalUrl){
```

```
    const hostname = '0.0.0.0';
    app.locals.url = externalUrl;
    app.locals.issuer = initIssuer(app.locals.url);
    app.listen(port, hostname, () => {
        console.log(`Server is running locally on http://${hostname}:${port}/ and from outside on ${externalUrl}`);
    });
} else {
    const os = require('os');
    const networkInterfaces = os.networkInterfaces();
    const hostname = networkInterfaces['wlo1'].filter((obj)=>obj['family']=='IPv4')[0]['address'];
    app.locals.url = `http://${hostname}:${port}`;
    app.locals.issuer = initIssuer(app.locals.url);
    app.listen(port, () => {
        console.log(`Server is running on local network on http://${hostname}:${port}/ `);
    });
}
```

Listing A.2: Sadržaj datoteke ./issuer/src/router.ts

```
import express from 'express';
import {offerPreauthController} from "./controllers/offer-preauth.controller";
import {offerAuthController} from "./controllers/offer-auth.controller";
import {authorizeController} from "./controllers/authorize.controller";
import {tokenController} from "./controllers/token.controller";
import {credentialController} from "./controllers/credential.controller";
const router = express.Router();
```

```
8
9  router.get('/.well-known/openid-credential-issuer', (req, res) => {
10     const issuer = req.app.locals.issuer;
11     const url = req.app.locals.url;
12     res.json({...issuer.issuerMetadata, authorization_endpoint: `${
       url}/authorize`});
13 });
14
15 router.post('/offer-preauth', offerPreauthController);
16
17 router.post('/offer-auth', offerAuthController);
18
19 router.get('/authorize', authorizeController);
20
21 router.post('/token', tokenController);
22
23 router.post('/credential',credentialController);
24
25 export default router;
```

Listing A.3: Sadržaj datoteke ./issuer/src/views/offer.ejs

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Credential offer</title>
6      <link rel="stylesheet" href="/index.css">
7  </head>
8  <body>
9      <a href="/">
10         <header>
11             <h1>Issuer</h1>
```

```
        </header>
    </a>
    <div>
        <h2>Created credential offer</h2>
        <h3>Qr code</h3>
        <img src="<%=qrCodeDataUri%>">
        <p><a href="<%=uri%>">Open from wallet</a></p>
        <h3>URI</h3>
        <p><%=uri%></p>
    </div>

</body>
</html>
```

Listing A.4: Sadržaj datoteke ./issuer/src/public/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Issuer</title>
    <link rel="stylesheet" href="/index.css">
</head>
<body>
<main>
    <header>
        <h1>Issuer</h1>
    </header>
    <div>
        <h2>Create credential offer (Pre Auth)</h2>
        <p>Use form below to create credential offer for COVID
   passport
            which will contain manufacturer of COVID vaccine
```

```
17              and last 2h. Select desired manufacturer.
18          </p>
19          <form action="/offer-preauth" method="post">
20              <fieldset>
21                  <legend>Select manufacturer:</legend>
22                  <label>
23                      <input type="radio" id="blue" name="manufacturer
    " value="Blue Inc." checked>
24                      Blue Inc.
25                  </label>
26                  <label>
27                      <input type="radio" id="red" name="manufacturer"
     value="Red Inc.">
28                      Red Inc.
29                  </label>
30              </fieldset>
31              <label for="email">Email</label>
32              <input type="email" id="email" name="email">
33              <button type="submit">Create offer</button>
34          </form>
35      </div>
36      <div>
37          <h2>Create credential offer (Auth)</h2>
38          <p>Use form below to create credential offer for COVID
    passport
39              which will contain manufacturer of COVID vaccine
40              and last 2h. Manufacturer will be provided based
41              on existing data. User with username:user and
42              password:user will ge Blue Inc. manufacturer.
43          </p>
44          <form action="/offer-auth" method="post">
45              <button type="submit">Create offer</button>
```

```
        </form>
    </div>
    <div>
        <h2>Metadata</h2>
        <p>Check <a href="/.well-known/openid-credential-issuer">
    metadata</a></p>
    </div>
</main>
</body>
</html>
```

Listing A.5: Sadržaj datoteke ./issuer/src/public/index.css

```
button {
    padding: 8px 20px;
    margin: 10px;
    font-size: 1em;
    background-color: #4285f4;
    color: #ffffff;
    border: none;
    border-radius: 10px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

button:hover {
    background-color: #3272bf;
}

button:active {
    transform: scale(0.95);
}

```

```
21 div, header {
22     background-color: #fff;
23     box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
24     border-radius: 8px;
25     padding: 20px;
26     margin: 20px auto;
27     width: 600px;
28 }
29
30 header:hover {
31     box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
32     transform: translateY(-2px);
33     transition: all 0.3s ease;
34 }
35
36 p {
37     overflow: auto;
38 }
39
40 a:has(header), a:has(header):visited, a:has(header):hover, a:has(
       header):active {
41     text-decoration: none;
42     color: inherit;
43 }
44
45 fieldset {
46     display: flex;
47     flex-direction: column;
48 }
49
50 input[type="radio"] {
51     margin-top: -1px;
```

```
    vertical-align: middle;
}

label {
    font-size: 1.25rem;
}
```

Listing A.6: Sadržaj datoteke ./issuer/src/utils/token.ts

```
/*
    This code is copied from @sphereon/oid4vci-issuer library
    Im using it to add auth code
 */

import {
    AccessTokenRequest,
    AccessTokenResponse,
    Alg,
    CNonceState,
    CredentialOfferSession,
    EXPIRED_PRE_AUTHORIZED_CODE,
    GrantTypes,
    INVALID_PRE_AUTHORIZED_CODE,
    IssueStatus,
    IStateManager,
    Jwt,
    JWTSignerCallback,
    PIN_NOT_MATCH_ERROR,
    PIN_VALIDATION_ERROR,
    PRE_AUTH_CODE_LITERAL,
    PRE_AUTHORIZED_CODE_REQUIRED_ERROR,
    TokenError,
    TokenErrorResponse,
```

```
25      UNSUPPORTED_GRANT_TYPE_ERROR,
26      USER_PIN_NOT_REQUIRED_ERROR,
27      USER_PIN_REQUIRED_ERROR,
28  } from '@sphereon/oid4vci-common'
29  import {v4} from 'uuid'
30
31  import {isPreAuthorizedCodeExpired} from '@sphereon/oid4vci-issuer'
32  import {base64url} from "jose";
33  import * as crypto from "node:crypto";
34
35  export interface ITokenEndpointOpts {
36      tokenEndpointDisabled?: boolean // Disable if used in an
        existing OAuth2/OIDC environment and have the AS handle tokens
37      tokenPath?: string // token path can either be defined here, or
        will be deduced from issuer metadata
38      interval?: number
39      cNonceExpiresIn?: number
40      tokenExpiresIn?: number
41      preAuthorizedCodeExpirationDuration?: number
42      accessTokenSignerCallback?: JWTSignerCallback
43      accessTokenIssuer?: string
44  }
45
46  export const generateAccessToken = async (
47      opts: Required<Pick<ITokenEndpointOpts, '
        accessTokenSignerCallback' | 'tokenExpiresIn' | '
        accessTokenIssuer'>> & {
48          preAuthorizedCode?: string,
49          code?: string,
50          alg?: Alg
51      },
52  ): Promise<string> => {
```

```
53      const { accessTokenIssuer, alg, accessTokenSignerCallback,
    tokenExpiresIn, preAuthorizedCode, code } = opts
54      // JWT uses seconds for iat and exp
55      const iat = new Date().getTime() / 1000
56      const exp = iat + tokenExpiresIn
57      const jwt: Jwt = {
58          header: { typ: 'JWT', alg: alg ?? Alg.ES256K },
59          payload: {
60              iat,
61              exp,
62              iss: accessTokenIssuer,
63              ...(preAuthorizedCode && { preAuthorizedCode }),
64              ...(code && {code})
65          },
66      }
67      return await accessTokenSignerCallback(jwt)
68  }
69
70  export const isValidGrant = (assertedState: CredentialOfferSession,
    grantType: string, codeVerifier?: string): boolean => {
71      if (assertedState.credentialOffer?.credential_offer?.grants) {
72          return (
73              Object.keys(assertedState.credentialOffer?.
    credential_offer?.grants).includes(GrantTypes.PRE_AUTHORIZED_CODE
    ) &&
74              grantType === GrantTypes.PRE_AUTHORIZED_CODE ||
75              // my code
76              Object.keys(assertedState.credentialOffer?.
    credential_offer?.grants).includes(GrantTypes.AUTHORIZATION_CODE)
      &&
77              grantType === GrantTypes.AUTHORIZATION_CODE &&
78                  /*
```

```
79                  rfc7636 says If the values are not
80                     equal, an error response indicating "
    invalid_grant" as described in
81                     Section 5.2 of [RFC6749] MUST be returned.
82                     BASE64URL-ENCODE(SHA256(ASCII(code_verifier))) ==
     code_challenge
83                  */
84              // @ts-ignore
85              assertedState.code_challange == base64url.encode(crypto.
    createHash('sha256').update(codeVerifier).digest())
86
87          )
88      }
89      return false
90  }
91
92  export const assertValidAccessTokenRequest = async (
93      request: Omit<AccessTokenRequest, "pre-authorized_code">,
94      opts: {
95          credentialOfferSessions: IStateManager<
    CredentialOfferSession>
96          expirationDuration: number
97      },
98  ) => {
99      const { credentialOfferSessions, expirationDuration } = opts
100     // Only pre-auth supported for now
101     if (request.grant_type !== GrantTypes.PRE_AUTHORIZED_CODE &&
    request.grant_type !== GrantTypes.AUTHORIZATION_CODE) {
102         throw new TokenError(400, TokenErrorResponse.invalid_grant,
    UNSUPPORTED_GRANT_TYPE_ERROR)
103     }
104
```

```
105     /*
106         error cases
107         also look at RFC 6749 OAuth 2.0
108     */
109
110     // Pre-auth flow
111     if(request.grant_type === GrantTypes.PRE_AUTHORIZED_CODE) {
112         if (!request[PRE_AUTH_CODE_LITERAL]) {
113             throw new TokenError(400, TokenErrorResponse.
    invalid_request, PRE_AUTHORIZED_CODE_REQUIRED_ERROR)
114         }
115
116         const credentialOfferSession = await credentialOfferSessions
    .getAsserted(request[PRE_AUTH_CODE_LITERAL])
117         credentialOfferSession.status = IssueStatus.
    ACCESS_TOKEN_REQUESTED
118         credentialOfferSession.lastUpdatedAt = +new Date()
119         await credentialOfferSessions.set(request[
    PRE_AUTH_CODE_LITERAL], credentialOfferSession)
120         if (!isValidGrant(credentialOfferSession, request.grant_type
    )) {
121             throw new TokenError(400, TokenErrorResponse.
    invalid_grant, UNSUPPORTED_GRANT_TYPE_ERROR)
122         }
123         /*
124         invalid_request:
125         the Authorization Server expects a PIN in the pre-authorized
     flow but the client does not provide a PIN
126          */
127         if (credentialOfferSession.credentialOffer.credential_offer
    ?.grants?.[GrantTypes.PRE_AUTHORIZED_CODE]?.user_pin_required &&
    !request.user_pin) {
```

```
128             throw new TokenError(400, TokenErrorResponse.
    invalid_request, USER_PIN_REQUIRED_ERROR)
129         }
130         /*
131         invalid_request:
132         the Authorization Server does not expect a PIN in the pre-
    authorized flow but the client provides a PIN
133          */
134         if (!credentialOfferSession.credentialOffer.credential_offer
    ?.grants?.[GrantTypes.PRE_AUTHORIZED_CODE]?.user_pin_required &&
    request.user_pin) {
135             throw new TokenError(400, TokenErrorResponse.
    invalid_request, USER_PIN_NOT_REQUIRED_ERROR)
136         }
137         /*
138         invalid_grant:
139         the Authorization Server expects a PIN in the pre-authorized
     flow but the client provides the wrong PIN
140         the End-User provides the wrong Pre-Authorized Code or the
    Pre-Authorized Code has expired
141          */
142         if (request.user_pin && !/[0-9{,8}]/.test(request.user_pin))
     {
143             throw new TokenError(400, TokenErrorResponse.
    invalid_grant, PIN_VALIDATION_ERROR)
144         } else if (request.user_pin !== credentialOfferSession.
    userPin) {
145             throw new TokenError(400, TokenErrorResponse.
    invalid_grant, PIN_NOT_MATCH_ERROR)
146         } else if (isPreAuthorizedCodeExpired(credentialOfferSession
    , expirationDuration)) {
147             throw new TokenError(400, TokenErrorResponse.
```

```
invalid_grant, EXPIRED_PRE_AUTHORIZED_CODE)
148         } else if (
149             request[PRE_AUTH_CODE_LITERAL] !==
150             credentialOfferSession.credentialOffer?.credential_offer
?.grants?.[GrantTypes.PRE_AUTHORIZED_CODE]?.[
PRE_AUTH_CODE_LITERAL]
151         ) {
152             throw new TokenError(400, TokenErrorResponse.
invalid_grant, INVALID_PRE_AUTHORIZED_CODE)
153         }
154         return {preAuthSession: credentialOfferSession}
155     }
156     // my code
157     // Auth code flow
158     /*
159         invalid_request rfc6749
160         The request is missing a required parameter other than grant
 type
161      */
162     if(!request["code"] && !request["code_verifier"] && !request["
redirect_uri"]
163         && !request["client_id"]){
164         throw new TokenError(400, TokenErrorResponse.invalid_request
, "Missing parametars")
165     }
166     const credentialOfferSession = await credentialOfferSessions.
getAsserted(request["code"])
167     credentialOfferSession.status = IssueStatus.
ACCESS_TOKEN_REQUESTED;
168     credentialOfferSession.lastUpdatedAt = +new Date();
169     await credentialOfferSessions.set(request["code"],
credentialOfferSession)
```

```
170     /*
171         invalid_grant
172      */
173     if (!isValidGrant(credentialOfferSession, request.grant_type,
    request["code_verifier"])) {
174         throw new TokenError(400, TokenErrorResponse.invalid_grant,
    UNSUPPORTED_GRANT_TYPE_ERROR)
175     }
176     return {authSession: credentialOfferSession}
177 }
178
179 export const createAccessTokenResponse = async (
180     request: AccessTokenRequest,
181     opts: {
182         credentialOfferSessions: IStateManager<
    CredentialOfferSession>
183         cNonces: IStateManager<CNonceState>
184         cNonce?: string
185         cNonceExpiresIn?: number // expiration in seconds
186         tokenExpiresIn: number // expiration in seconds
187         // preAuthorizedCodeExpirationDuration?: number
188         accessTokenSignerCallback: JWTSignerCallback
189         accessTokenIssuer: string
190         interval?: number
191     },
192 ) => {
193     const { credentialOfferSessions, cNonces, cNonceExpiresIn,
    tokenExpiresIn, accessTokenIssuer, accessTokenSignerCallback,
    interval } = opts
194
195     if(request.grant_type === GrantTypes.PRE_AUTHORIZED_CODE) {
196         const preAuthorizedCode = request[PRE_AUTH_CODE_LITERAL] as
```

```
string;

        const cNonce = opts.cNonce ?? v4()
        await cNonces.set(cNonce, {cNonce, createdAt: +new Date(),
preAuthorizedCode})

        const access_token = await generateAccessToken({
            tokenExpiresIn,
            accessTokenSignerCallback,
            preAuthorizedCode,
            accessTokenIssuer,
        })
        const response: AccessTokenResponse = {
            access_token,
            token_type: 'bearer',
            expires_in: tokenExpiresIn,
            c_nonce: cNonce,
            c_nonce_expires_in: cNonceExpiresIn,
            authorization_pending: false,
            interval,
        }
        const credentialOfferSession = await credentialOfferSessions
.getAsserted(preAuthorizedCode)
        credentialOfferSession.status = IssueStatus.
ACCESS_TOKEN_CREATED
        credentialOfferSession.lastUpdatedAt = +new Date()
        await credentialOfferSessions.set(preAuthorizedCode,
credentialOfferSession)
        return response
    }
    //auth code
    const code = request["code"] as string;
```

```
224     const credentialOfferSession = await credentialOfferSessions.
    getAsserted(code);
225     const issuerState = credentialOfferSession.issuerState;
226
227     const cNonce = opts.cNonce ?? v4()
228     await cNonces.set(cNonce, {cNonce, createdAt: +new Date(),
    issuerState});
229
230     const access_token = await generateAccessToken({
231         tokenExpiresIn,
232         accessTokenSignerCallback,
233         //preAuthorizedCode,
234         code,
235         accessTokenIssuer,
236     })
237     const response: AccessTokenResponse = {
238         access_token,
239         token_type: 'bearer',
240         expires_in: tokenExpiresIn,
241         c_nonce: cNonce,
242         c_nonce_expires_in: cNonceExpiresIn,
243         authorization_pending: false,
244         interval,
245     }
246     credentialOfferSession.status = IssueStatus.ACCESS_TOKEN_CREATED
247     credentialOfferSession.lastUpdatedAt = +new Date()
248     // @ts-ignore
249     await credentialOfferSessions.set(credentialOfferSession.
    issuerState, credentialOfferSession)
250     //await credentialOfferSessions.delete(code);
251     return response
252 }
```

Listing A.7: Sadržaj datoteke ./issuer/src/controllers/authorize.controller.ts

```
import {VcIssuer} from "@sphereon/oid4vci-issuer";
import {AuthorizationRequest} from "@sphereon/oid4vci-common";
import {randomBytes} from "node:crypto";

export const authorizeController = async (req, res) => {
    const issuer = req.app.locals.issuer as VcIssuer<object>;
    const authReq = req.query as unknown as AuthorizationRequest;
    console.log('auth req', JSON.stringify(authReq, null, 2));

    // invalid_request rfc7636
    if(!authReq["code_challenge"] &&
        !authReq["code_challenge_method"] &&
        authReq["code_challenge_method"] !== "S256"){
        res.status(400).json({error: "invalid_request"});
        return;
    }

    /*
        This is out of scope of specs OID4VCI, OID Connect and OAuth
    2.0
        Implementation using hardcoded username and password and
        use of HTTP Basic is chosen because of simplicity.
 */
    const b64auth = (req.headers.authorization || '').split(' ')[1]
   || ''
    const [login, password] = Buffer.from(b64auth, 'base64').
   toString().split(':')
    if(login != 'user' || password != 'user'){
        res.setHeader('WWW-Authenticate', 'Basic realm="Issuer"')
            .sendStatus(401);
        return;
```

```
    }

    const data = {manufacturer:"Blue Inc."};

    const codeLengthInBytes = !!parseInt(process.env.CODE_LENGTH) ?
    parseInt(process.env.CODE_LENGTH) : 16;
    const code: string = randomBytes(codeLengthInBytes).toString("
    hex");

    let offerSession = await issuer.credentialOfferSessions.get(
    authReq.issuer_state);
    await issuer.credentialOfferSessions.delete(authReq.issuer_state
    );
    await issuer.credentialOfferSessions.set(code, {...offerSession,
        credentialDataSupplierInput: data,
        // @ts-ignore
        code_challange: authReq.code_challenge,
        code_challenge_method:authReq.code_challenge_method,
        code
    });

    console.log(`redirect with code ${code}`)
    res.redirect(`${authReq.redirect_uri}/?code=${code}`);
}
```

Listing A.8: Sadržaj datoteke ./issuer/src/controllers/credential.controller.ts

```
import {VcIssuer} from "@sphereon/oid4vci-issuer";
import {importJWK, jwtVerify, KeyLike} from "jose";
import {UniResolver} from "@sphereon/did-uni-client";
import {CredentialRequestV1_0_11} from "@sphereon/oid4vci-common";
import {credentialDataSupplier} from "../service/issuer";

```

```
/*
    Checks if credential has required fields
 */
function validateCredentialReq(credentialReq:
   CredentialRequestV1_0_11) {
    const requiredProperties = ['type', 'format', 'proof'];
    const keys = Object.keys(credentialReq);
    if (!requiredProperties.every(prop => keys.includes(prop))) {
        return false;
    }
    if (keys.length !== requiredProperties.length) {
        return false;
    }
    return true;
}

export const credentialController = async (req, res) => {
    const issuer = req.app.locals.issuer as VcIssuer<object>;

    // resolve issuer public key
    let publicKey: Uint8Array | KeyLike;
    try{
        const resolver = new UniResolver();
        const didResolutionResult = await resolver.resolve(process.
   env.PUBLIC_KEY_DID);
        const {publicKeyJwk} = didResolutionResult.didDocument.
   verificationMethod[0];
        publicKey = await importJWK(publicKeyJwk);
    } catch (e) {
        console.error(e);
        res.setHeader('Cache-Control', 'no-store').status(500)
            .json({error: "server_error"})
```

```
36      }
37
38      // check if token is valid
39      const authorisationHeader = req.headers["authorization"];
40      if(!authorisationHeader && !req.headers["authorization"].
    startsWith("Bearer")){
41          /*
42             https://datatracker.ietf.org/doc/html/rfc6750#section-3.1
43             If the request lacks any authentication information (e.g
    ., the client
44             was unaware that authentication is necessary or attempted
     using an
45             unsupported authentication method), the resource server
    SHOULD NOT
46             include an error code or other error information.
47           */
48          console.error("Auth headers non existant or malformed")
49          res.sendStatus(401);
50          return;
51      }
52      try {
53          const token = authorisationHeader.replace("Bearer ", "");
54          const {payload} = await jwtVerify(token, publicKey);
55          const code = !!payload.preAuthorizedCode ?
56              payload.preAuthorizedCode as string :
57              payload.code as string;
58
59          if(Date.now()/1000 > payload.exp){
60              throw new Error(‘Token expired‘);
61          }
62          if(payload.iss !== issuer.issuerMetadata.credential_issuer){
63              throw new Error("Invalid iss in token");
```

```
64          }
65          const offer = await issuer.credentialOfferSessions.get(code)
    ;
66          if(!offer){
67              throw new Error("Offer does not exist means that
    preAuthCode or code is invalid");
68          }
69
70      } catch (e){
71          console.error(e)
72          res.status(401).json({error:"invalid_token"});
73          return;
74      }
75
76      const credentialReq = req.body as CredentialRequestV1_0_11;
77      console.log('credential req', JSON.stringify(credentialReq, null
    , 2))
78
79      // error cases
80      // check credential request if it has required fields
81      if( validateCredentialReq(credentialReq) ){
82          res.setHeader('Cache-Control','no-store').status(400)
83              .json({error:"invalid_credential_request"})
84          return;
85      }
86      // check format, req format needs to be supported
87      if( !issuer.issuerMetadata.credentials_supported.some(
    credentialSupported => {
88          return credentialSupported.format === credentialReq.format
89      })){
90          res.setHeader('Cache-Control','no-store').status(400)
91              .json({error:"unsupported_credential_format"})
```

```
92          return;
93      }
94      // check type, every type need to be supported
95      if(issuer.issuerMetadata.credentials_supported.some(
    credentialSupported => {
96          return credentialReq["types"].every((type:string) => {
97              return type in credentialSupported["types"]
98          })
99      })){
100         res.setHeader('Cache-Control','no-store').status(400)
101             .json({error:"unsupported_credential_type"})
102         return;
103     }
104     // check encryption params
105     // this demo doesnt encrypt credential request
106
107     // https://openid.net/specs/openid-4-verifiable-credential-
    issuance-1_0.html#name-credential-issuer-provided-
108
109     try {
110         const credentialRes = await issuer.issueCredential({
111             credentialRequest: credentialReq,
112             credentialDataSupplier: credentialDataSupplier,
113             cNonceExpiresIn: issuer.cNonceExpiresIn,
114             tokenExpiresIn: 10 * 60 * 1000, //needs to be same as in
     /token
115         });
116         console.log('credential res', JSON.stringify(credentialRes,
    null, 2));
117         res.json(credentialRes);
118     } catch (e){
119         console.error(e);
```

```
120         res.setHeader('Cache-Control','no-store').status(500)
121             .json({error:"server_error"})
122     }
123 }
```

Listing A.9: Sadržaj datoteke ./issuer/src/controllers/offer-preauth.controller.ts

```
1  import {VcIssuer} from "@sphereon/oid4vci-issuer";
2  import validator from "validator";
3  import {randomBytes} from "node:crypto";
4  import {CredentialOfferSession, IssueStatus} from "@sphereon/oid4vci
       -common/dist/types/StateManager.types";
5  import {sendEmail} from "../service/sendEmail";
6
7  export const offerPreauthController = async (req,res) => {
8      const issuer = req.app.locals.issuer as VcIssuer<object>;
9      const data = req.body;
10
11     if(!('manufacturer' in data) ||
12         !('email' in data) ||
13         Object.keys(data).length !== 2 ||
14         !validator.isEmail(data.email) ||
15         (data.manufacturer !== "Blue Inc." && data.manufacturer !==
       "Red Inc.")){
16         res.status(400).send('Error 400')
17         return;
18     }
19
20     const codeLengthInBytes = !!parseInt(process.env.CODE_LENGTH) ?
       parseInt(process.env.CODE_LENGTH) : 16;
21     const code: string = randomBytes(codeLengthInBytes).toString("
       hex");
22
```

```
23      const pinLength = 4;
24      const pad: string = "0".repeat(pinLength);
25      let pin: string = (randomBytes(32).readUInt32BE() % 10**
    pinLength).toString();
26      pin = pad.substring(0, pad.length - pin.length) + pin;
27
28      const grants = {
29          'urn:ietf:params:oauth:grant-type:pre-authorized_code': {
30              'pre-authorized_code': code,
31              user_pin_required: true,
32          },
33      };
34
35      const offerResult = await issuer.createCredentialOfferURI({
36          grants,
37          credentials: ['covid-passport'],
38          qrCodeOpts: {
39              size: 400 // it will not work if it is small
40          },
41          pinLength: pinLength
42      })
43
44      const session: CredentialOfferSession = {
45          createdAt: Date.now(),
46          credentialOffer: {
47              credential_offer: {
48                  credential_issuer: req.app.locals.issuer.
    credential_issuer,
49                  credentials: ['jwt_vc'],
50                  grants
51              }
52          },
```

```
53          credentialDataSupplierInput: data,
54          userPin: pin,     // only when pin is required
55          status: IssueStatus.OFFER_CREATED,
56          lastUpdatedAt: Date.now(),
57          preAuthorizedCode: code
58      };
59      await issuer.credentialOfferSessions.set(code,session);
60
61      try{
62          await sendEmail(data.email, pin);
63      } catch (e) {
64          res.status(500).send(‘Email could not be sent!‘);
65          console.error(e);
66          return;
67      }
68
69      res.render(’offer’,offerResult);
70  }
```

Listing A.10: Sadržaj datoteke ./issuer/src/controllers/offer-auth.controller.ts

```
1  import {VcIssuer} from "@sphereon/oid4vci-issuer";
2  import {randomBytes} from "node:crypto";
3  import {CredentialOfferSession, IssueStatus} from "@sphereon/oid4vci
       -common/dist/types/StateManager.types";
4
5  export const offerAuthController = async (req,res) => {
6      const issuer = req.app.locals.issuer as VcIssuer<object>;
7
8      const issuerState: string = randomBytes(32).toString("hex");
9
10     const grants = {
11         authorization_code: {
```

```
            issuer_state: issuerState
        }
    };

    const offerResult = await issuer.createCredentialOfferURI({
        grants,
        credentials: ['covid-passport'],
        qrCodeOpts: {
            size: 400 // it will not work if it is small
        }
    })

    const session: CredentialOfferSession = {
        createdAt: Date.now(),
        credentialOffer: {
            credential_offer: {
                credential_issuer: req.app.locals.issuer.
    credential_issuer,
                credentials: ['jwt_vc'],
                grants
            }

        },
        status: IssueStatus.OFFER_CREATED,
        lastUpdatedAt: Date.now(),
        issuerState: issuerState,
    };
    await issuer.credentialOfferSessions.set(issuerState,session);

    res.render('offer',offerResult);
}
```

Listing A.11: Sadržaj datoteke ./issuer/src/controllers/token.controller.ts

```
import {VcIssuer} from "@sphereon/oid4vci-issuer";
import {AccessTokenRequest, Jwt, TokenError} from "@sphereon/oid4vci-common";
import {importPKCS8, SignJWT} from "jose";
import {assertValidAccessTokenRequest, createAccessTokenResponse} from "../utils/token";

export const tokenController = async (req, res) => {
    const issuer = req.app.locals.issuer as VcIssuer<object>;
    const tokenReq = req.body as AccessTokenRequest;
    console.log('token req', JSON.stringify(tokenReq, null, 2));

    const accessTokenSignerCallback = async (jwt: Jwt): Promise<string> => {
        const pemPrivateKey =  Buffer.from(process.env.PRIVATE_KEY , 'base64').toString('ascii');
        const privateKey = await importPKCS8(pemPrivateKey, 'ES256');
        return new SignJWT(jwt.payload)
            .setProtectedHeader({alg:'ES256'})
            .setExpirationTime('30s') // anything above 5m is considered to be long lived
            .sign(privateKey)
    };
    try {
        await assertValidAccessTokenRequest(tokenReq, {
            credentialOfferSessions: issuer.credentialOfferSessions,
            expirationDuration: 10 * 60 * 1000
        });
        const tokenRes = await createAccessTokenResponse(tokenReq, {
            credentialOfferSessions: issuer.credentialOfferSessions,
```

```
            cNonces: issuer.cNonces,
            cNonceExpiresIn: issuer.cNonceExpiresIn,
            tokenExpiresIn: 10 * 60 * 1000,
            accessTokenSignerCallback,
            accessTokenIssuer: issuer.issuerMetadata.
    credential_issuer,
            interval: 3000
        });
        console.log('token res', JSON.stringify(tokenRes, null, 2));
        res.setHeader('Cache-Control', 'no-store').json(tokenRes);
    } catch (e){
        console.error(e);
        if(e instanceof TokenError){
            res.setHeader('Cache-Control','no-store').status(e.
    statusCode)
                .json({error:e.responseError});
            return;
        }
        res.setHeader('Cache-Control','no-store').status(500)
            .json({error:"server_error"});
    }
}
```

Listing A.12: Sadržaj datoteke ./issuer/src/service/signCredential.ts

```
import {importPKCS8, SignJWT} from "jose";
import {ICredentialSubject} from "@sphereon/ssi-types/dist/types/w3c
    -vc";

export const signCredentialCallback = async (opts) => {
    console.log(`debug signing credential >${JSON.stringify(opts,
    null, 4)}<`)
    const {
```

```
7           credential,
8           credentialRequest,
9           format,
10          jwtVerifyResult
11      } = opts;
12      const payload = {
13          vc: credential,
14      }
15      const header = {
16          alg: "ES256",
17          typ: "JWT"
18      }
19
20      const pemPrivateKey =  Buffer.from(process.env.PRIVATE_KEY , '
    base64').toString('ascii');
21      const privateKey = await importPKCS8(pemPrivateKey, 'ES256');
22      // @ts-ignore
23      const credentialJwt: string = await new SignJWT(payload)
24          .setProtectedHeader(header)
25          .setIssuer(credential.issuer as string)
26          .setNotBefore( (new Date(credential.issuanceDate as string))
    .getTime()/1000 )
27          .setExpirationTime( (new Date(credential.expirationDate as
    string)).getTime()/1000 )
28          .setSubject((credential.credentialSubject as
    ICredentialSubject).id)
29          .sign(privateKey);
30      return new Promise((resolve, reject) => {
31          resolve(credentialJwt)
32      })
33  }
```

Listing A.13: Sadržaj datoteke ./issuer/src/service/jwtVerify.ts

```
import {decodeProtectedHeader, importJWK} from "jose";
import {UniResolver} from "@sphereon/did-uni-client";
import * as common from "@sphereon/oid4vci-common";
import {jwtVerify} from "jose";

export const jwtVerifyCallback = async ({jwt, kid}) => {
    try {
        const decodedJwt = decodeProtectedHeader(jwt);
        // resolving did
        const resolver = new UniResolver();
        const didResolutionResult = await resolver.resolve(kid ===
    undefined ? decodedJwt.kid : kid);
        const {publicKeyJwk} = didResolutionResult.didDocument.
    verificationMethod[0];
        const publicKey = await importJWK(publicKeyJwk);
        // verify jwt
        const jwtObj = await jwtVerify(jwt, publicKey);
        console.log(`debug jwt decoded >${JSON.stringify(jwtObj)}<`)
    ;
        const header: common.JWTHeader = jwtObj.protectedHeader;
        const payload: common.JWTPayload = jwtObj.payload;
        const jwtVerifyResult = {
            jwt: {
                header,
                payload
            },
            alg: header.alg,
            didDocument: didResolutionResult.didDocument,
            did: JSON.stringify(didResolutionResult.didDocument)
        }
        return new Promise((resolve, reject) => {
```

```
            resolve(jwtVerifyResult)
        });
    } catch (e) {
        return new Promise((resolve, reject) => {
            reject(e)
        });
    }
}
```

Listing A.14: Sadržaj datoteke ./issuer/src/service/issuer.ts

```
import {
    CredentialDataSupplier, CredentialDataSupplierResult,
   CredentialSignerCallback,
    CredentialSupportedBuilderV1_11,
    VcIssuer,
    VcIssuerBuilder
} from "@sphereon/oid4vci-issuer";
import jwtlib from "jsonwebtoken";
import {jwtVerifyCallback} from "./jwtVerify";
import {JWTVerifyCallback} from "@sphereon/oid4vci-common";
import {signCredentialCallback} from "./signCredential";

const credentialSupported = new CredentialSupportedBuilderV1_11()
    .withId('covid-passport')
    .withFormat('jwt_vc_json')
    .withTypes(["VerifiableCredential"])
    .withCredentialSupportedDisplay({
        name:"Covid Passort",
        locale:"en-US"
    })
    .build();

```

```
22 export const initIssuer = (url: string): VcIssuer<any> => {
23     return new VcIssuerBuilder()
24         .withAuthorizationServer(url)
25         .withCredentialEndpoint(`${url}/credential`)
26         .withCredentialIssuer(url)
27         .withIssuerDisplay({
28             name: 'Default issuer display',
29             locale: 'en-US',
30         })
31         .withInMemoryCredentialOfferState()
32         .withInMemoryCNonceState()
33         .withCredentialsSupported(credentialSupported)
34         .withJWTVerifyCallback(jwtVerifyCallback as
    JWTVerifyCallback<any>)
35         .withCredentialSignerCallback(signCredentialCallback as
    CredentialSignerCallback<any>)
36         .build()
37 }
38
39 export const credentialDataSupplier: CredentialDataSupplier = (args)
     => {
40     console.log(args);
41     const {credentialDataSupplierInput} = args;
42     const payload = jwtlib.decode(args.credentialRequest.proof.jwt)
    as jwtlib.JwtPayload;
43     const startTime = new Date();
44     const endTime = new Date(startTime.getTime() + 2*60*60*1000); //
     2 hours from startTime
45     const credential = {
46         '@context': [
47             "https://www.w3.org/2018/credentials/v1"
48         ],
```

```
        type: ['VerifiableCredential'],
        issuer: process.env.PUBLIC_KEY_DID,
        issuanceDate: startTime.toJSON(),
        expirationDate: endTime.toJSON(),
        credentialSubject: {
            "id": payload.iss,
            "manufacturer": credentialDataSupplierInput.manufacturer
    ,
        }
    };
    const result: CredentialDataSupplierResult = {
        credential
    };
    return new Promise((resolve, reject) => resolve(result));
}
```

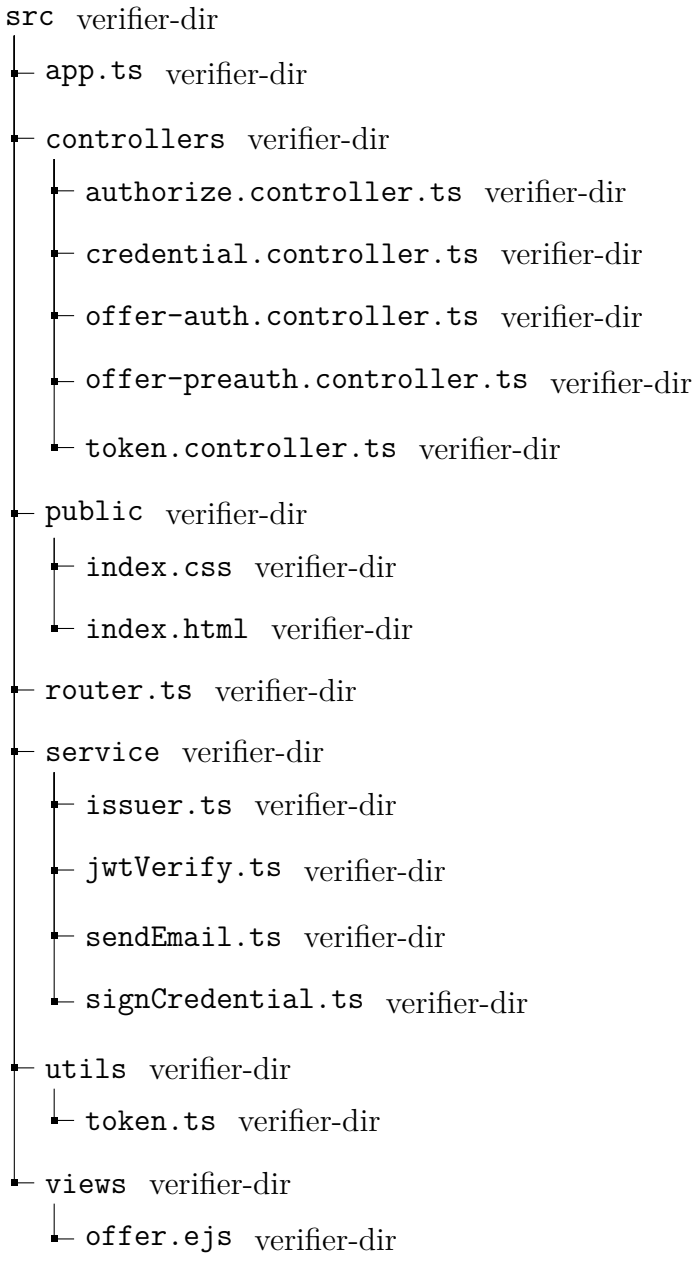
Listing A.15: Sadržaj datoteke ./issuer/src/service/sendEmail.ts

```
import dotenv from "dotenv";
import nodemailer from "nodemailer";

export async function sendEmail(destination: string, pin: string){
    let transporter = nodemailer.createTransport({
        service: "Outlook365",
        host: "outlook.office365.com",
        port: 993,
        auth: {
            user: process.env.EMAIL_USER,
            pass: process.env.EMAIL_PASS
        },
        tls: {
            ciphers:'SSLv3'
        }
```

```
16      });
17
18      return await transporter.sendMail({
19          from: process.env.EMAIL_USER,
20          to: destination,
21          subject: 'Pin for https://openid4vc-issuer.onrender.com',
22          text: `Your pin is ${pin}`
23      })
24  }
```

Privitak B: Struktura direktorija koda Issuer aplikacije



Privitak C: Kod Verifier aplikacije

Listing C.1: Sadržaj datoteke ./verifier/src/app.ts

```
import dotenv from 'dotenv';
import express from 'express';
import path from "node:path";
import indexRouter from "./router";
import {initRp} from "./services/verifier";
import {CapabilityUrlsManager} from "./utils/CapabilityUrlsManager";
dotenv.config();

const externalUrl = process.env.RENDER_EXTERNAL_URL;
const port = process.env.PORT ? parseInt(process.env.PORT) : 3000;

const app = express();

app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(express.static(path.join(__dirname, 'public')));
app.set('view engine','ejs');
app.set('views', path.join(__dirname, 'views'));

app.use('/', indexRouter);

app.locals.capabilityUrlsManager = new CapabilityUrlsManager<string
    >();

```

```
24 if(externalUrl){
25     const hostname = '0.0.0.0';
26     app.locals.url = externalUrl;
27     app.locals.rp = initRp(app.locals.url);
28     app.listen(port, hostname, () => {
29         console.log(`Server is running locally on http://${hostname
       }:${port}/ and from outside on ${externalUrl}`);
30     });
31 } else {
32     const os = require('os');
33     const networkInterfaces = os.networkInterfaces();
34     const hostname = networkInterfaces['wlo1'].filter((obj)=>obj['
       family']=='IPv4')[0]['address'];
35     app.locals.url = `http://${hostname}:${port}`;
36     app.locals.rp = initRp(app.locals.url);
37     app.listen(port, () => {
38         console.log(`Server is running on local network on http://${
       hostname}:${port}/ `);
39     });
40 }
```

Listing C.2: Sadržaj datoteke ./verifier/src/router.ts

```
1 import express from 'express';
2 import {createAuthReq} from "./controllers/createAuthReq.controller"
      ;
3 import {authRequest} from "./controllers/authReq.controller";
4 import {submitPresentation} from "./controllers/submitPresentation.
      contoller";
5 import {authStatus} from "./controllers/authStatus.controller";
6
7 const router = express.Router();
8
```

```
router.post('/create-auth-req', createAuthReq);

//this url should expire and aybe use something else instead of
    corrId https://www.rfc-editor.org/rfc/rfc9101.html
/*
    This url needs to be capability url. Look up
    https://www.rfc-editor.org/rfc/rfc9101.html#name-uri-referencing
    -the-request and
    https://www.rfc-editor.org/rfc/rfc6819#section-5.1.4.2.2 and
    https://www.w3.org/TR/capability-urls/
 */
router.get('/auth-req/:reference', authRequest);

router.post('/post', submitPresentation);

/*
    Out od scope of specification of protocol OID4VP
 */
router.post('/auth-status', authStatus);
export default router;
```

Listing C.3: Sadržaj datoteke ./verifier/src/views/example.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Verifier</title>
    <link rel="stylesheet" href="/index.css">
</head>
<body>
<main>
    <a href="/">
```

```
        <header>
            <h1>Verifier</h1>
        </header>
    </a>

    <div correlationId="<%=correlationId%>" id="presentation">
        <h2>Created authorization request details</h2>
        <h3>QR Code</h3>
        <img src="<%=qrCodeDataUri%>">
        <p><a href="<%=authReqUri%>">Open from wallet</a></p>

        <h3>URI</h3>
        <p id="req-uri"><%=authReqUri%></p>

        <h3>Authorisation request</h3>
        <p>Here is authorization request obtained from request_uri
and decoded from JWT</p>
        <pre id="authReqObj"></pre>
    </div>
    <div id="status">
        <h2>Status of created presentation</h2>
        <p>Status of presentation is <strong><span id="status-text"
></span></strong></p>
    </div>
    <div id="passport" hidden>
        <h2>Covid passport</h2>
        <p>Manufacturer is <span id="passport-manufacturer"></span
></p>
        <p>Not valid before <span id="passport-nbf"></span></p>
        <p>Valid until <span id="passport-exp"></span></p>
        <p>Current valid status is <strong><span id="passport-status
"></span></strong></p>
```

```
39
40          <br>
41          <p>Verifier has id token with payload</p>
42          <pre id="id-token"></pre>
43          <p>Verifier has vp token with payload</p>
44          <pre id="vp-token"></pre>
45      </div>
46  </main>
47  <script src="/authStatus.js"></script>
48  </body>
```

Listing C.4: Sadržaj datoteke ./verifier/src/public/authStatus.js

```
1  let idToken;
2  let vpToken;
3  let intervals = [];
4  function showAuthReqObj(){
5      const xhr = new XMLHttpRequest();
6      const uri = document.querySelector('#req-uri').innerHTML;
7      const uriDecoded = decodeURIComponent(uri);
8      const params = {};
9      for(let param of uriDecoded.split('?')[1].split('&')){
10         console.log(param)
11         const key = param.split('=')[0];
12         const value = param.split('=')[1];
13         params[key] = value;
14     }
15     console.log(params);
16     xhr.open('GET',params['request_uri']);
17     xhr.onload = function(){
18         const token = this.response;
19         const base64Url = token.split('.')[1];
20         const base64 = base64Url.replace(/-/g, '+').replace(/_/g, '/
```

```
    ');
21          const jsonPayloadString = decodeURIComponent(window.atob(
    base64).split('').map(function(c) {
22              return '%' + ('00' + c.charCodeAt(0).toString(16)).slice
    (-2);
23          }).join(''));
24          const parsed = JSON.parse(jsonPayloadString);
25          document.querySelector('#authReqObj').innerHTML = JSON.
    stringify(parsed, null, 2);
26      }
27      xhr.send();
28  }
29
30  function check(){
31      const xhr = new XMLHttpRequest();
32      const correlationId = document.querySelector('#presentation').
    getAttribute('correlationId');
33      xhr.open('POST',`/auth-status`);
34      xhr.setRequestHeader("Content-Type", "application/json;charset=
    UTF-8");
35      xhr.onload = function(){
36          if (this.status !== 200){
37              console.log('Status is not 200');
38              return;
39          }
40          const response = JSON.parse(this.response);
41          console.log(response)
42          document.querySelector('#status-text').innerHTML = response.
    status;
43          if(response.status !== 'verified'){
44              return;
45          }
```

```
46          const {data} = response;
47          document.querySelector('#passport-nbf').innerHTML = (new
    Date(data.nbf * 1000)).toUTCString();
48          document.querySelector('#passport-exp').innerHTML = (new
    Date(data.exp * 1000)).toUTCString();
49          const statusHtml = document.querySelector('#passport-status'
    );
50          if(new Date() < new Date(data.exp * 1000)){
51              statusHtml.innerHTML = 'VALID';
52              statusHtml.style.color = 'green';
53          } else {
54              statusHtml.innerHTML = 'NOT VALID';
55              statusHtml.style.color = 'red';
56          }
57          document.querySelector('#passport-manufacturer').innerHTML =
     data.manufacturer;
58          document.querySelector('#vp-token').innerHTML = JSON.
    stringify(response.vpTokenPayload, null, 2);
59          document.querySelector('#id-token').innerHTML = JSON.
    stringify(response.idTokenPayload, null, 2);
60          vpToken = response.vpToken;
61          idToken = response.idToken;
62
63          document.querySelector('#passport').removeAttribute("hidden"
    );
64          document.querySelector('#presentation').setAttribute("hidden
    ","")
65      }
66      xhr.send(JSON.stringify({correlationId}));
67  }
68  intervals.push(setInterval(check,1000));
69  showAuthReqObj();
```

Listing C.5: Sadržaj datoteke ./verifier/src/public/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Verifier</title>
    <link rel="stylesheet" href="/index.css">
</head>
<body>
<main>
    <header>
        <h1>Verifier</h1>
    </header>
    <div>
        <h2>Create authorization request</h2>
        <p>This demo is for presentation purposes.</p>
        <p>Use form below to create authorization request
            for checking COVID Passport manufacturer.</p>
        <form method="post" action="/create-auth-req">
            <button type="submit">Create authorization request</
   button>
        </form>
    </div>
</main>
</body>
```

Listing C.6: Sadržaj datoteke ./verifier/src/public/index.css

```
button {
    padding: 8px 20px;
    margin: 10px;
    font-size: 1em;
    background-color: #4285f4;
```

```
    color: #ffffff;
    border: none;
    border-radius: 10px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

button:hover {
    background-color: #3272bf;
}

button:active {
    transform: scale(0.95);
}

div, header {
    background-color: #fff;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    border-radius: 8px;
    padding: 20px;
    margin: 20px auto;
    width: 600px;
}

header:hover {
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
    transform: translateY(-2px);
    transition: all 0.3s ease;
}

p {
    overflow: auto;
```

```
}

a:has(header), a:has(header):visited, a:has(header):hover, a:has(
    header):active {
    text-decoration: none;
    color: inherit;
}

pre {
    overflow: auto;
}
```

Listing C.7: Sadržaj datoteke ./verifier/src/utils/CapabilityUrlsManager.ts

```
type CapabilityUrlManagerItem<Type> = {
    createdAt: number,
    item: Type
};

export class CapabilityUrlsManager<Type>{
    map: Map<string, CapabilityUrlManagerItem<Type>>
    expiresIn: number;

    constructor(){
        this.map = new Map<string, CapabilityUrlManagerItem<Type>>;
        this.expiresIn = 5 * 60 * 1000; // 5 minutes
    }

    set(url: string, item:Type){
        this.map.set(url, {createdAt: Date.now(), item})
    }

    get(url: string){
```

```
        if(!this.map.has(url)){
            return null;
        }
        const {createdAt, item} = this.map.get(url);
        if(createdAt + this.expiresIn < Date.now()){
            this.map.delete(url);
            return null;
        }
        return item;
    }
}
```

Listing C.8: Sadržaj datoteke ./verifier/src/controllers/authStatus.controller.ts

```
import {RP} from "@sphereon/did-auth-siop";
import {decodeJwt} from "jose";

export const authStatus = async (req,res) => {
    const rp = req.app.locals.rp as RP;
    const { correlationId } = req.body;
    const authReqState = await rp.sessionManager.
    getRequestStateByCorrelationId(correlationId);
    if(authReqState === undefined){
        res.sendStatus(404);
        return;
    }
    const authResState = await rp.sessionManager.
    getResponseStateByCorrelationId(correlationId);
    if(authResState == undefined){
        res.json(authReqState);
        return;
    }

```

```
18      //app specific
19      const vpToken = authResState.response.payload.vp_token.toString
        ();
20      const vpTokenPayload = decodeJwt(vpToken);
21      const idToken = authResState.response.payload.id_token.toString
        ();
22      const idTokenPayload= decodeJwt(idToken);
23      const credential = decodeJwt(vpTokenPayload['vp']['
        verifiableCredential'][0])
24
25      if(authResState.status !== 'verified'){
26          res.json(authResState)
27          return;
28      }
29
30      const data = {
31          nbf: credential.nbf,
32          exp: credential.exp,
33          manufacturer: credential['vc']['credentialSubject']['
        manufacturer']
34      }
35      res.json({...authResState, data, vpTokenPayload, vpToken,
        idTokenPayload, idToken})
36  }
```

Listing C.9: Sadržaj datoteke ./verifier/src/controllers/submitPresentation.contoller.ts

```
1  import {AuthorizationResponsePayload, RP} from "@sphereon/did-auth-
       siop";
2  import {presentationDefinition} from "../services/verifier";
3  import {PresentationDefinitionLocation} from "@sphereon/did-auth-
       siop/dist/authorization-response/types";
4
```

```
export const submitPresentation = async (req,res) => {
    const rp = req.app.locals.rp as RP;
    const authRes = req.body as AuthorizationResponsePayload;
    console.log(authRes)
    try {
        const verifiedAuthRes = await rp.verifyAuthorizationResponse
    (authRes, {
            presentationDefinitions: [{
                definition: presentationDefinition,
                location: PresentationDefinitionLocation.
    CLAIMS_VP_TOKEN
            }]
        });
        console.log('Presented', verifiedAuthRes.
    authorizationResponse.payload);
    } catch (e){
        console.error(`Error while posting presentation ${e}`);
    }
    /*
        If the Response Endpoint has successfully processed the
    Authorization Response or
        Authorization Error Response, it MUST respond with HTTP
    status code 200.
        https://openid.github.io/OpenID4VP/openid-4-verifiable-
    presentations-wg-draft.html#section-6.2-17
      */
    res.sendStatus(200);
}
```

Listing C.10: Sadržaj datoteke ./verifier/src/controllers/authReq.controller.ts

```
import {RP} from "@sphereon/did-auth-siop";
import {CapabilityUrlsManager} from "../utils/CapabilityUrlsManager"
```

```
    ;

export const authRequest = async (req, res) => {
    const rp = req.app.locals.rp as RP;
    const capabilityUrlsManager = req.app.locals.
    capabilityUrlsManager as CapabilityUrlsManager<string>;
    const reference = req.params.reference;

    const correlationId = capabilityUrlsManager.get(reference);
    if(correlationId === null){
        res.sendStatus(404);
        return;
    }

    const authReq = await rp.sessionManager.
    getRequestStateByCorrelationId(correlationId);
    res.send(await authReq.request.requestObject.toJwt());
}
```

Listing C.11: Sadržaj datoteke ./verifier/src/controllers/createAuthReq.controller.ts

```
import {RP, SupportedVersion} from "@sphereon/did-auth-siop";
import {CapabilityUrlsManager} from "../utils/CapabilityUrlsManager"
    ;
import {randomBytes} from "node:crypto";
import QRCode from "qrcode";

export const createAuthReq = async (req, res) => {
    const rp = req.app.locals.rp as RP;
    const url = req.app.locals.url;
    const capabilityUrlsManager = req.app.locals.
    capabilityUrlsManager as CapabilityUrlsManager<string>;
    const correlationId: string = randomBytes(32).toString("hex");
```

```
    const nonce: string = randomBytes(32).toString("hex");
    const state: string = randomBytes(32).toString("hex");
    const reference: string = randomBytes(32).toString("hex");
    capabilityUrlsManager.set(reference, correlationId);
    const authReq = await rp.createAuthorizationRequest({
        correlationId, // if there is no corrId it uses state as corrId
        nonce,
        state,
        responseURI: `${url}/post`,
        responseURIType: "response_uri",
        version: SupportedVersion.JWT_VC_PRESENTATION_PROFILE_v1,
        requestByReferenceURI: `${url}/auth-req/${reference}`
    });
    const authReqUri = (await authReq.uri()).encodedUri;
    const qrCodeDataUri = await QRCode.toDataURL(authReqUri);

    console.log(authReq)
    res.render('example.ejs',{authReqUri, qrCodeDataUri, correlationId});
}
```

Listing C.12: Sadržaj datoteke ./verifier/src/services/verifier.ts

```
import {IPresentationDefinition} from "@sphereon/pex/dist/main/lib/types/Internal.types";
import {
    InMemoryRPSessionManager,
    PassBy, PresentationVerificationCallback,
    PropertyTarget,
    ResponseIss,
    ResponseMode,
    ResponseType,
```

```
    RevocationVerification,
    RP, Scope,
    SigningAlgo, SubjectType,
    SupportedVersion
} from "@sphereon/did-auth-siop";
import EventEmitter from "node:events";
import {presentationVerificationCallback} from "./
   presentationVerificationCallback";

export const presentationDefinition: IPresentationDefinition ={
    id: "presentation-definition-id",
    name: "Covid passpoer manufacturer name",
    purpose: "Identify manufacturer of covid vaccinee",
    input_descriptors: [
        {
            "id": "covid-passport-valid-pd",
            "name": "Covid passport manufacturer name",
            "purpose": "To see is covid passport manufacturer",
            "constraints": {
                "fields": [
                    {
                        "path": [
                            "$.vc.credentialSubject.manufacturer",
                            "$.credentialSubject.manufacturer"
                        ]
                    }
                ]
            }
        }
    ]
}

```

```
40 const eventEmiter = new EventEmitter();
41
42
43 export const initRp = (url:string ):RP => {
44     const rpKeys = {
45         hexPrivateKey: process.env.PRIVATE_KEY_HEX,
46         did: process.env.PUBLIC_KEY_DID,
47         didKey: process.env.PUBLIC_KEY_DID,
48         alg: SigningAlgo.ES256
49     }
50
51     return RP.builder()
52         .withRequestBy(PassBy.REFERENCE, `${url}/auth-req`)
53         .withResponseMode(ResponseMode.POST)
54         .withInternalSignature(rpKeys.hexPrivateKey, rpKeys.did,
    rpKeys.didKey, rpKeys.alg)
55         .withSignature(rpKeys)
56         .withIssuer(ResponseIss.JWT_VC_PRESENTATION_V1)
57         .withPresentationVerification(
    presentationVerificationCallback as
    PresentationVerificationCallback)
58         .withRevocationVerification(RevocationVerification.NEVER)
59         .withSupportedVersions(SupportedVersion.
    JWT_VC_PRESENTATION_PROFILE_v1)
60         .withResponseType(ResponseType.ID_TOKEN)
61         .withClientMetadata({
62             subject_syntax_types_supported: ['did', 'did:web', 'did:
    jwk'],
63             idTokenSigningAlgValuesSupported: [SigningAlgo.ES256],
64             requestObjectSigningAlgValuesSupported: [SigningAlgo.
    ES256],
65             responseTypesSupported: [ResponseType.ID_TOKEN],
```

```
            vpFormatsSupported: {jwt_vc: {alg: [SigningAlgo.ES256]}},
            scopesSupported: [Scope.OPENID_DIDAUTHN, Scope.OPENID],
            subjectTypesSupported: [SubjectType.PAIRWISE],
            subjectSyntaxTypesSupported: ['did', 'did:web', 'did:jwk'],
            passBy: PassBy.VALUE
        })
        .withPresentationDefinition({
            definition: presentationDefinition,
        }, PropertyTarget.REQUEST_OBJECT)
        .withClientId(`${url}`)
        .withSessionManager(new InMemoryRPSessionManager(eventEmiter))
        .withEventEmitter(eventEmiter)
        .build()
}
```

Listing C.13: Sadržaj datoteke ./verifier/src/services/presentationVerificationCallback.ts

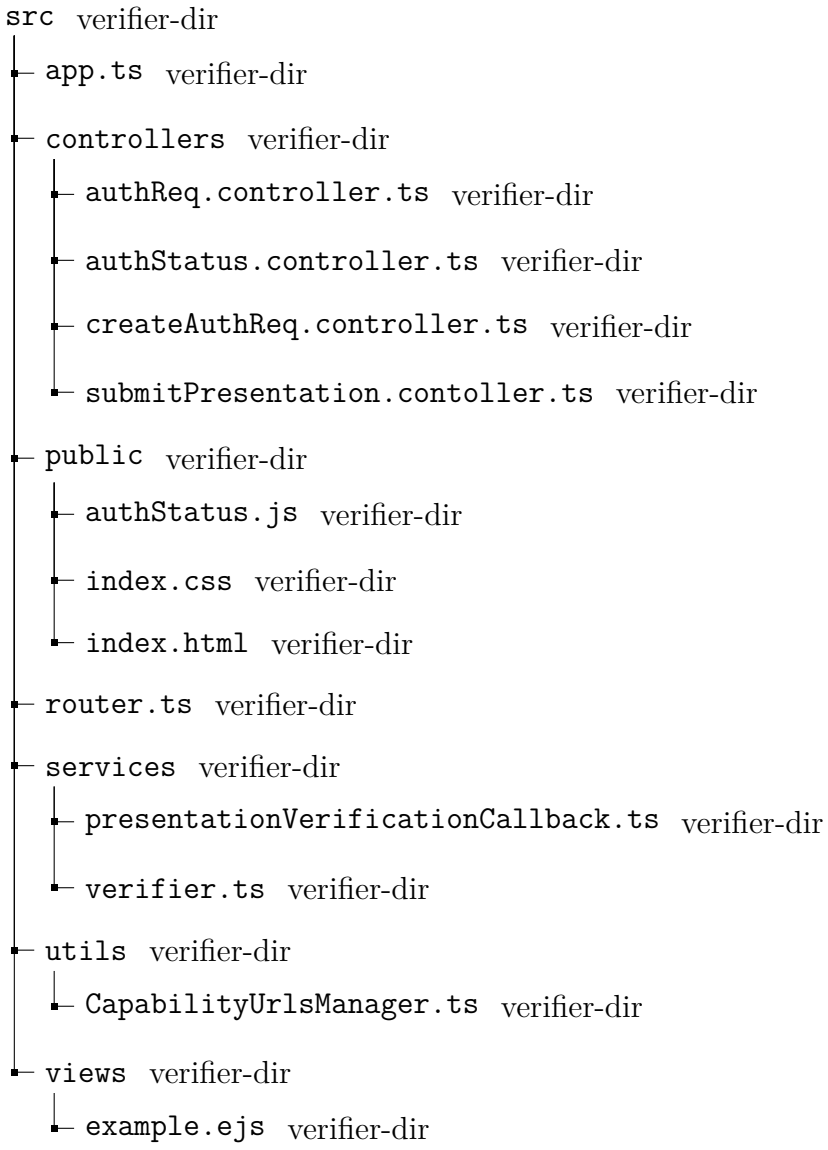
```
import {parseJWT, PresentationVerificationResult} from "@sphereon/did-auth-siop";
import {PEX} from "@sphereon/pex";
import {UniResolver} from "@sphereon/did-uni-client";
import {compactVerify, importJWK} from "jose";
import jp from "jsonpath";
import {presentationDefinition} from "./verifier";

export const presentationVerificationCallback = async (args, presentationSubmission)=> {
  let result: PresentationVerificationResult;
  try {
```

```
11    if (typeof args !== "string") {
12      throw new Error("App only supports presentations encoded in
      JWT")
13    }
14    const pex = new PEX();
15    const resolver = new UniResolver();
16
17    const presentationDecoded = parseJWT(args);
18    const walletDid = await resolver.resolve(presentationDecoded.
      payload.iss);
19    const walletKey = await importJWK(walletDid.didDocument.
      verificationMethod[0].publicKeyJwk);
20    const verificationResult = await compactVerify(args, walletKey);
      // throws error if signature is not right
21
22    const pexResult = pex.evaluatePresentation(
      presentationDefinition, args)
23    if(pexResult.errors.length > 0){
24      throw new Error(`Error during evaluating presentation ${
      pexResult.errors.toString()}`)
25    }
26    const credentials = []
27    for(let descriptor of pexResult.value.descriptor_map){
28      const selectedCredentials = jp.query(presentationDecoded.
      payload.vp, descriptor.path);
29      selectedCredentials.forEach((credential) => credentials.push(
      credential))
30    }
31    for(let credential of credentials){
32      const credentialDecoded = parseJWT(credential);
33      const issuerDid = await resolver.resolve(credentialDecoded.
      payload.iss);
```

```
34        const issuerKey = await importJWK(issuerDid.didDocument.
    verificationMethod[0].publicKeyJwk)
35        const verificationResult = await compactVerify(credential,
    issuerKey); // throws error if signature is not right
36      }
37      result = {verified:true};
38    } catch(err) {
39      result = {verified:false, reason:err};
40    }
41    return new Promise((resolve, reject) => {resolve(result)});
42  }
```

Privitak D: Struktura direktorija koda Verifier aplikacije



Privitak E: Kod pomoćnih skripti

caption,

```
const fs = require('fs');
const jose = require('jose');
const path = require('path');
const {base64ToHexString} = require("@sphereon/did-auth-siop");

const generateKeys = async (alg) => {
    console.log(`Generating keys using alg:${alg}`)
    const {privateKey, publicKey} = await jose.generateKeyPair(alg);
    const publicPem = await jose.exportSPKI(publicKey);
    const privatePem = await jose.exportPKCS8(privateKey);
    const privatePemB64 = Buffer.from(privatePem).toString('base64')
    ;
    const publicJwk = JSON.stringify(await jose.exportJWK(publicKey)
    ,null,2);
    const privateJwk = JSON.stringify(await jose.exportJWK(
    privateKey),null,2);
    const privateKeyHex = base64ToHexString((await jose.exportJWK(
    privateKey)).d, 'base64url');

    const dir = './keys';
    if (!fs.existsSync(dir)){
        fs.mkdirSync(dir);
    }

```

```
21      fs.writeFileSync(path.join(process.cwd(), `./keys/public.pem`),
    publicPem);
22      fs.writeFileSync(path.join(process.cwd(), `./keys/private.pem`),
     privatePem);
23      /*
24          Put in ENV variable PRIVATE_KEY then use as
25          const key = Buffer.from(process.env.PRIVATE_KEY , 'base64').
    toString('ascii');
26      */
27      fs.writeFileSync(path.join(process.cwd(), `./keys/private-pem-
    b64.txt`), privatePemB64);
28      /*
29          Publish this inside DID
30       */
31      fs.writeFileSync(path.join(process.cwd(), `./keys/public.jwk.
    json`),publicJwk);
32      fs.writeFileSync(path.join(process.cwd(), `./keys/private.jwk.
    json`),privateJwk);
33      /*
34          Put in ENV variable PRIVATE_KEY_HEX
35       */
36      fs.writeFileSync(path.join(process.cwd(), `./keys/private-hex.
    txt`),privateKeyHex);
37  };
38
39  generateKeys('ES256')
40  .then(()=>console.log('Keys generated'))
41  .catch((e)=>console.log(e))
```

Privitak F: Primjer objavljenih DID dokumenata

Listing F.1: DID dokument dobiven razlučivanjem "did:web:lovremitrovic.github.io:did-database:verifier"

```
{
    "@context": [
        "https://www.w3.org/ns/did/v1",
        "https://w3id.org/security/suites/jws-2020/v1"
    ],
    "id": "did:web:lovremitrovic.github.io:did-database:verifier",
    "verificationMethod": [
        {
            "id": "did:web:alovremitrovic.github.io:did-database:verifier",
            "type": "JsonWebKey2020",
            "controller": "did:web:lovremitrovic.github.io:did-database:verifier",
            "publicKeyJwk": {
              "kty": "EC",
              "x": "Ka9DEDwNjlgo-GMnAhK4BwDWC5-Mqd5GFK0QRaDE_wA",
              "y": "XbdNETxysZtUOITsKEaP6woTCL0Tc-1p5Pp6ngnst6s",
              "crv": "P-256"
            }
        }
    ]
}
```

Listing F.2: DID dokument dobiven razlučivanjem "did:web:lovremitrovic.github.io:did-database:issuer"

```
{
    "@context": [
        "https://www.w3.org/ns/did/v1",
        "https://w3id.org/security/suites/jws-2020/v1"
    ],
    "id": "did:web:LovreMitrovic.github.io:did-database:issuer",
    "verificationMethod": [
        {
            "id": "did:web:LovreMitrovic.github.io:did-database:issuer",
            "type": "JsonWebKey2020",
            "controller": "did:web:LovreMitrovic.github.io:did-database:issuer",
            "publicKeyJwk": {
                  "kty": "EC",
                  "x": "6v0wx7K7CegMKNzggIlkmok7KLgwbxXScI86emXsTH8",
                  "y": "yPBO_uMtUMLTKkkh7NxcwY3JR_fgE6d2VT1N6cxaRp8",
                  "crv": "P-256"
               }
        }
    ]
}
```

Privitak G: Tamarin model

Listing G.1: Model protokola OID4VC i OID4VP

```
theory OpenID4VCI
begin

builtins: hashing, asymmetric-encryption, signing

restriction Equality: "All x y #i. Eq(x,y) @i ==> x = y"
restriction OneKeyPairPerEntity: "All A #i #j. RegisterKeyPair(A)@i
    & RegisterKeyPair(A)@j ==> #i = #j"

// Register key
rule Register_pk:
    [ Fr(~ltk) ]
    --[RegisterKeyPair($A)]→
    [ !Ltk($A, ~ltk), !Pk($A, pk(~ltk)) ]

// Get key
// Adversary can always get any public key
rule Get_pk:
    [ !Pk(A, pubkey) ]
    -→
    [ Out(pubkey) ]

rule Reveal_ltk:
    [ !Ltk(A, ltk) ]
```

```
    --[ LtkReveal(A) ]→
    [ Out(ltk) ]

/*
    Modeling secure channal
*/

rule ChanOut_S:
        [ Out_S($A,$B,x) ]
        --[ ChanOut_S($A,$B,x) ]→
        [ !Sec($A,$B,x) ]

rule ChanIn_S:
        [ !Sec($A,$B,x) ]
      --[ ChanIn_S($A,$B,x) ]→
        [ In_S($A,$B,x) ]


// User enters claims in browser
// Browser sends POST request to issuer vis TLS
rule User_create_offer:
    [ ]
    --[User_created_offer($claims)]→
    [ Out_S($user, $issuer, <'create_offer',$claims>)]


// Init credential offer on issuer
// and send it to wallet e.g. scanning qr code
rule Issuer_credential_offer:
    let credential_offer = <~code, $issuer> in
    [ In_S($user, $issuer, <'create_offer',$claims>), Fr(~code)]
    --[Secret(~code)]→
```

```
56      [ Out_S($issuer, $wallet, <'credential_offer',credential_offer>)
        , St_iss_1($issuer,~code, $claims)]
57
58
59  // Wallet consumes offer and requests token
60  rule Wallet_token_req:
61      let credential_offer = <~code, $issuer>
62          token_req = ~code //token_req = <pkWal, ~code>
63      in
64      [ In_S($issuer, $wallet, <'credential_offer',credential_offer>)
        ]
65      -->
66      [ Out_S($wallet, $issuer, <'token_req',token_req>), St_wal_1(
        $wallet, ~code)]
67
68  rule Issuer_token_res:
69      let
70          sig = sign(~code, ~ltkIss)
71          token = <sig,~code>
72          token_req = ~code//<~client_id, ~code>
73          token_res = <token, ~c_nounce>
74      in
75      [ In_S($wallet, $issuer, <'token_req',token_req>),
76          !Ltk($issuer, ~ltkIss),
77          Fr(~c_nounce), St_iss_1($issuer, ~code, $claims) ]
78      --[Secret(token), Secret(~c_nounce)]->
79      [ Out_S($issuer, $wallet, <'token_res',token_res>), St_iss_2(
        $issuer, ~code, $claims, ~c_nounce) ]
80
81  rule Wallet_credential_request:
82      let
83          token_res = <token, ~c_nounce>
```

```
84          code = snd(token)
85          sig = sign(~c_nounce, ~ltkWal)
86          proof = <sig, ~c_nounce>
87          credential_req = <token, proof>
88      in
89      [In_S($issuer, $wallet,<'token_res',token_res>),
90          !Ltk($wallet,~ltkWal),
91          St_wal_1($wallet, ~codeSt)]
92
93      --[Eq(code, ~codeSt)]→
94      [Out_S($wallet, $issuer,<'credential_req',credential_req>),
    St_wal_2($wallet, token)]
95
96  rule Issuer_credential_response:
97      let
98          credential_req = <token, proof>
99
100         sig = sign(<pkIss, pkWal, $claims>, ~ltkIss)
101         credential = <sig, <pkIss, pkWal, $claims>>
102         credential_res = <credential, ~c_nounce>
103     in
104     [In_S($wallet, $issuer,<'credential_req',credential_req>),
105         St_iss_2($issuer, ~code, $claims, ~c_nounce),
106         !Ltk($issuer, ~ltkIss), !Pk($issuer, pkIss), !Pk($wallet,
    pkWal)]
107
108     // token is code signed by Iss
109     // proof is nounce signed by wall
110     --[Eq(verify(fst(token), ~code, pkIss), true),
111         Eq(verify(fst(proof), ~c_nounce, pkWal), true)]→
112
113     [Out_S($issuer, $wallet, <'credential_res',credential_res>)]
```

```
114
115     // At the end user checks credential and decides
116     // to accept it or not
117
118 rule Wallet_accept_credential:
119         let credential = <sig, pkIss, pkWal, claimsFromCredential>
120             credential_res = <credential, ~c_nounce>
121         in
122         [In_S($issuer, $wallet, <'credential_res',credential_res>),
123             !Ltk($wallet,~ltkWall), !Pk($wallet, pkWal),!Pk($issuer,
    pkIss)]
124
125         --[Eq(verify(fst(credential), snd(credential), pkIss), true)
    ]→
126
127         [Out_S($wallet, $user, <'credential',credential>)]
128
129 rule User_accept_credential:
130     let credential = <sig, pkIssFromCredential, pkWalFromCredential,
     claimsFromCredential>
131     in
132     [In_S($wallet, $user, <'credential',credential>)]
133     --[Credential_is_issued_as(pkIssFromCredential,
    pkWalFromCredential, claimsFromCredential)]→
134     [!Wallet_has(credential)]
135
136 rule User_Create_Presentation:
137     []
138     -→
139     [Out_S($user, $verifier, <'create_presentation',
    $presentationClaims>) ]
140
```

```
141 // verifier creates request, stores it, shows qr code containing
    path to wallet
142 rule Verifier_shows_path_to_auth_req:
143     [In_S($user, $verifier, <'create_presentation',$claims>),
144          Fr(~corrId), Fr(~state), Fr(~nounce), Fr(~reference)]
145     --[Secret(~corrId), Secret(~state), Secret(~nounce)]→
146     [!St_ver_store_reqest($presentationClaims, ~state, ~nounce, ~
    reference),
147         Out_S($verifier, $wallet, <'scan_presentation_uri',~
    reference>)//this is scanning qr code
148     ]
149
150 // wallet send GET request to reference url
151 rule Wallet_asks_for_auth_req:
152     [In_S($verifier, $wallet, <'scan_presentation_uri',~reference>)]
153     -→
154     [Out_S($wallet, $verifier, <'presentation_uri',~reference>)]//
    this is GET req via TLS
155
156 rule Verifier_sends_auth_req:
157     let sig = sign(<~nounce,~state>, ~ltkVer)
158         authReq = <sig, ~nounce, ~state>
159     in
160     [In_S($wallet, $verifier, <'presentation_uri',~reference>),
161     !St_ver_store_reqest($presentationClaims, ~state, ~nounce, ~
    reference),
162     !Ltk($verifier,~ltkVer)]
163     -→
164     [Out_S($verifier, $wallet, <'auth_req',authReq>)]
165
166 rule Wallet_sends_auth_res:
```

```
167     let credential = <sigCredential, pkIssFromCredential,
    pkWalFromCredential, claimsFromCredential>
168         authReq = <sigAuthReq, ~nounce, ~state>
169         sigAuthRes = sign(<~nounce, ~state, credential,pkWal>,~
    ltkWal)
170         authRes = <sigAuthRes,~nounce, ~state, credential,pkWal>
171     in
172     [In_S($verifier, $wallet, <'auth_req',authReq>),
173         !Wallet_has(credential),
174         !Pk($wallet, pkWal),!Pk($verifier, pkVer), !Ltk($wallet, ~
    ltkWal)]
175     --[/*provjeri je li auth req ispravno potpisan od verifiera*/
176         Eq(verify(fst(authReq), snd(authReq), pkVer), true)]->
177     [Out_S($wallet, $verifier, <'auth_res',authRes>)]
178
179 rule Verifier_consumes_response:
180     let credentialRecv = <sigCredentialRecv, pkIssFromCredentialRecv
    , pkWalFromCredentialRecv, claimsFromCredentialRecv>
181         authRes = <sigAuthRes,~nounce, ~state, credentialRecv,pkWal>
182     in
183     [In_S($wallet, $verifier, <'auth_res',authRes>),
184         !St_ver_store_reqest($presentationClaims, ~state, ~nounce, ~
    reference),
185         !Pk($wallet, pkWal),!Pk($issuer, pkIss)]
186
187     --[/*provjeri je li authRes potpisan od walleta*/
188         Eq(verify(fst(authRes), snd(authRes), pkWal), true),
189         /*provjeri je li credential potpisan od issuera*/
190         Eq(verify(fst(credentialRecv), snd(credentialRecv), pkIss),
    true),
191         Finish()]->
192
```

```
193     []
194
195
196 lemma executable:
197     exists-trace
198     " Ex #i. Finish()@i
199         & not (Ex b #k. LtkReveal(b)@k)"
200
201 lemma autentific_claims:
202     "All pkIss pkWal claimsFromCredential #i.
    Credential_is_issued_as(pkIss, pkWal, claimsFromCredential)@i
203     ==> Ex claims #j. User_created_offer(claims)@j
204     & j<i
205     & claims=claimsFromCredential"
206
207 lemma secret:
208     " All x #i. Secret(x)@i ==> not (Ex #j. K(x)@j) "
209 end
```