Provera identiteta

Autentifikacija ≠ identifikacija ≠ autorizacija

- autentifikacija (provera identiteta) = utvrđivanje identiteta korisnika
- identifikacija = utvrđivanje da li je korisnik poznat sistemu
- autorizacija = utvrđivanje prava koja korisnik ima nad resursima u sistemu

- autorizacija zahteva uspešnu autentifikaciju
  - autentifikacija prethodi autorizaciji
- autentifikacija podrazumeva identifikaciju
  - identifikacija je sastavni deo postupka autentifikacije

Učesnici u autentifikaciji

- onaj koji se predstavlja kao korisnik (**prover**, claimant)
- onaj koji proverava identitet (**verifier**, recipient)

Alati za autentifikaciju

- šifrovane poruke
  - ispravan šifrat potvrđuje autentičnost poruke i pošiljaoca
- checksum funkcije
  - izračunate nad porukom za autentifikaciju i tajnim ključem
- heš funkcije

Protokol za autentifikaciju

- proces u realnom vremenu kojim se utvrđuje identitet
- rezultat
  - korisnikov identitet je autentičan
  - korisnikov identitet nije autentičan

- karakteristike protokola za autentifikaciju
  - zanemarljiva verovatnoća da treći učesnik, predstavljajući se kao prover, može da navede verifier-a na pozitivan rezultat autentifikacije
  - verifier ne može da koristi informacije koje je dostavio prover kako bi se predstavio kao prover trećem učesniku

Sredstva za autentifikaciju

- korisnik se identifikuje pomoću određenog identifikatora (npr. username)
- svoj identitet potvrđuje pomoću više faktora
  - nečega što zna (lozinka, PIN, šifra sefa)
  - nečega što ima (smart kartica, ključ, generator vremenski zavisnog PIN-a)
  - nečega što je njegova karakteristika (otisak prsta, oblik lica, retina, prepoznavanje glasa)

- autentifikacija je pouzdanija ako se koristi više faktora
  - lozinka se može pogoditi
  - ključ se može izgubiti
  - prepoznavanje lica: "false positives" procenat je još uvek značajan
- banke koriste dvostruku autentifikaciju na bankomatima
  - traže i karticu i PIN

Klasifikacija šema za autentifikaciju

- slabe (weak) šeme
  - jednostavne za implementaciju
  - podložne napadima
  - lozinka, PIN
- jake (strong) šeme
  - zasnovane na challenge-response protokolu
  - obuhvataju kriptografske tehnike

Autentifikacija pomoću lozinke

- najraširenij metod za autentifikaciju
- lozinka ~ tajni ključ
- postupak
  - korisnik se prijavljuje pomoću para (korisničko ime, lozinka)
  - server proverava da li dati par postoji u registru postojećih korisnika
    - registar = sistemski fajl, zaštićen od čitanja

- ako se lozinka smešta direktno u sistemski fajl, nema zaštite od privilegovanih korisnika sistema ili čitanja fajla iz bekapa

- rešenje: ne čuva se lozinka, već njen heš
  - prilikom prijavljivanja izračunava se heš i poredi sa onim koji je skladišten u fajlu
  - otvoreni tekst lozinke nigde se ne skladišti trajno

Autentifikacija pomoću lozinke

- treba izbegavati skladištenje lozinke kao otvorenog teksta
  - upravo ono što radi Windows u registry-ju
    - "usluga" korisniku da ne mora u toku rada da ponovo unosi svoju lozinku prilikom prijavljivanja na druge sisteme u mreži
  - na UNIX/Linux sistemima, heševi lozinki se čuvaju u /etc/passwd ili /etc/shadow

Autentifikacija pomoću lozinke

- napad pomoću rečnika (dictionary attack)
  - izračunava se heš svake reči iz rečnika (recimo milion stavki)
  - porede se izračunati heševi iz rečnika sa sadržajima skladištenim u sistemskom fajlu

- otežati napad pomoću "salt" vrednosti
  - salt: slučajan niz znakova koji se dodaje na lozinku pre izračunavanja heša
  - salt i heš se smeštaju u fajl
  - napad na konkretnu lozinku traži heširanje kompletnog rečnika za dati salt
    - i tako za svaku lozinku posebno
    - umesto da se rečnik heširа jednom, mora se heširati za svaku salt vrednost

Autentifikacija pomoću lozinke

- jednokratne lozinke
  - prilikom svakog prijavljivanja koristi se različita lozinka
  -
- naredna lozinka se određuje pomoću
  - liste unapred definisanih lozinki
    - ne koriste se sekvencijalno već sistem traži *i*-tu lozinku iz liste
  - sekvencijalnog ažuriranja
    - nakon pozitivne autentifikacije korisnik će poslati lozinku za naredno prijavljivanje
    - prva lozinka je unapred poznata
  - sekvencijalnog izračunavanja
    - prva lozinka je unapred poznata
    - svaka sledeća lozinka je heš prethodne lozinke: h(h(...h(pwd)...))

Autentifikacija pomoću lozinke

- SKEY postupak
  - korisnik unese slučajan broj R
  - sistem generiše $f(R) = x_1$, $f(f(R)) = x_2$, ... $f(f(...f(R)...)) = x_{100}$ i daje niz $x_{1\text{-}100}$ korisniku, a skladišti i x_{101}
  - prilikom prvog prijavljivanja korisnik šalje x_{100}, server izračunava $f(x_{100})$ i poredi ga sa x_{101}
    - ako su jednaki, korisnik je pozitivno autentifikovan
  - sistem zamenjuje x_{101} sa x_{100}
  - korisnik za dalje prijavljivanje koristi poslednji neiskorišćeni broj iz niza

- svaki broj se koristi samo jednom, a f je jednosmerna funkcija
- napadač ako i pročita x_{101}, taj podatak mu ne znači puno

Autentifikacija pomoću PIN-a

- PIN = personal identification number
- PIN = kratak niz cifara (4 do 10)

- PIN se skladišti na fizičkom uređaju npr. kartici ili tokenu

- broj mogućih vrednosti ključa je mali → ograničava se broj pokušaja korišćenja PIN-a

Jake šeme za autentifikaciju

- challenge-response princip
  - korisnik potvrđuje svoj identitet tako što demonstrira poznavanje tajne informacije
    - ako poznaje tajnu informaciju, smatra se da je autentičan
  - pri tome se tajna informacija ne odaje sistemu

  - sistem korisniku šalje podatak koji se menja tokom vremena
  - može da uključuje
    - timestamp
    - random number
    - sequence number
  - korisnik izračunava odgovor na osnovu dobijenog podatka i tajne informacije
  - ako je odgovor ispravan, sistem smatra da je korisnik autentičan

Jake šeme za autentifikaciju

- realizacija challenge-response postupka pomoću
  - kriptosistema sa tajnim ključem
  - kriptosistema sa javnim ključem
  - zero-knowledge tehnika

Challenge-response sa tajnim ključem

- korisnik i sistem dele tajni ključ i unapred dogovoreni simetrični algoritam
- postupak
  1. sistem šalje korisniku slučajan broj *r*
  2. korisnik šifruje *r* tajnim ključem i šalje ga sistemu
  3. sistem dešifruje *r* i proverava da li je jednak originalnom

- broj *r* koji se šifruje može se dopuniti proizvoljnom porukom *m*, kako bi se sprečili replay napadi
- ako je dodatna poruka dugačka, može se šifrovati heš od [*r*, *m*]

Challenge-response sa tajnim ključem

- moguća je i obostrana autentifikacija
  1. sistem šalje korisniku slučajan broj *r*
  2. korisnik šalje poruku koja se sastoji iz šifrovanog broja *r* (tajnim ključem) i novog slučajnog broja *r'*
     - ako se dešifrovanjem dobije originalni broj *r* korisnik je autentičan
  3. sistem šalje korisniku šifrat broja *r'*
     - ako se dešifrovanjem dobije originalni broj *r'* sistem je autentičan

Challenge-response sa javnim ključem

- postupak
  1. sistem:
     - generiše slučajan broj *r*
     - računa njegov heš h(*r*)
     - šifruje *r* i proizvoljnu poruku *m* korisnikovim javnim ključem c(*r*, *m*, *pk*)
     - šalje korisniku [h(*r*), c(*r*, *m*, *pk*)]
  2. korisnik:
     - dešifruje šifrovani deo poruke (time dobija *r*)
     - izračunava svoj heš h(*r*) i poredi ga sa primljenim
     - ako su jednaki, šalje *r* sistemu
  3. sistem proverava da li je primljeni r jednak originalnom

Challenge-response sa javnim ključem

- jedan pristup za mitigaciju za *password sniffing* je upravo korišćenje javnog ključa servera za šifrovanje bilo kog prenosa informacija vezanih za lozinku na server
  - samo server može da dešifruje informacije koristeći svoj privatni ključ
  - Secure Shell (SSH) i Secure Sockets Layer (SSL) ovo rade
    - Server-side mode zahteva da server ima sertifikat
    - Client-side mode zahteva da i klijent ima sertifikat
- alternativni pristup je izbegavanje prenošenja lozinke i korišćenje protokola koji zahteva samo poznavanje lozinke da bi se uspešno pokrenuo (Kerberos)

Challenge-response sa zero-knowledge tehnikom

- prethodni postupci mogu da odaju deo tajne informacije
  - ako se napadač predstavi kao sistem i šalje specifične challenge vrednosti pomoću kojih će doći do tajne informacije

- zero-knowlegde postupci bi trebalo da eliminišu ovu mogućnost tako što sistemu ne otkrivaju nikakvu informaciju

Fiat-Shamir

- postavka (Trusted Third Party)
  1. izaberu se dva velika prosta broja p, q i izračuna n = p·q
  2. brojevi p, q ostaju tajni; objavljuje se samo n
  3. prover (dokazivač) bira tajni ključ $s \in Z_n^*$, gcd(s, n) = 1
  4. prover računa javni ključ (verifikacioni parametar) $v = s^2 \bmod n$
  5. prover registruje (n, v) kod verifikatora (ili u javnom direktorijumu)

Fiat-Shamir

- interaktivni protokol (ponavlja se t puta da bi se verovatnoća prevare svela na 2^{-t})
- ponavljajući koraci protokola (i = 1,…,t):
  a. commit (obavezivanje)
     - prover bira slučajni $r_i \in Z_n^*$
     - izračuna $x_i = r_i^2 \bmod n$ i šalje x_i verifikatoru
  b. challenge (izazov)
     - verifikator bira nasumičan bit $c_i \in \{0, 1\}$ i vraća ga proveru
  c. response (odgovor)
     - prover računa $y_i = r_i \cdot s^{ci} \bmod n$
     - prover šalje y_i verifikatoru
  d. verify (provera)
     a. verifikator proverava $y_i^2 \bmod n = (x^i \cdot v^{ci}) \bmod n$
     b. ako je provera ispravna, sledeća runda; inače se odmah odbacuje

Fiat-Shamir - osobine

- soundness
  - garantuje da lažni prover (koji ne zna tajni s) ne može da prevari verifikatora sa verovatnoćom većom od 2^{-t}
  - računa se tako da je jedna runda uspešna za prevaranta sa verovatnoćom najviše ½
    - posle t nezavisnih rundi: Pr[prevariti sve runde] ≤ $(½)^t = 2^{-t}$
    - znači što više rundi → eksponencijalno manja šansa za uspeh prevaranta
- zero-knowledge
  - garantuje da verifikator ne uči ništa o tajnom s osim da prover zaista zna s
  - simulator:
    - postoji algoritam koji, bez poznavanja s, može da proizvede niz poruka (x_i, c_i, y_i) koji je statistički neodvojiv od stvarnog protokola
    - verifikator ne može da razlikuje "pravi" tok interakcije od onog koje je generisao simulator
  - implikacija:
    - ni posmatranjem svih poruka, verifikator ne dobija nikakav dodatni uvid u s

Napadi na mehanizme za autentifikaciju

- napadi sa lažnim predstavljanjem (*impersonation attacks*)
  - napadač dolazi u posed tajne informacije (lozinka, PIN, tajni ključ) i koristi je za autentifikaciju

- napadi sa ponovljenim porukama (*replay attacks*)
  - napadač prisluškuje komunikaciju u toku autentifikacije i ponavlja je
    - npr. ako nalog za plaćanje potpiše pravi korisnik banke, a napadač ponavo šalje istu poruku banci (time ponavlja plaćanje)
  - anti-replay mere: timestamp, slučajan broj, itd.

- napadi sa uvođenjem kašnjenja (*forced delay attacks*)
  - napadač presretne poruku, sačeka određeni period vremena, i prosledi je na odredište
    - korisnik dobije timeout, misli da je transakcija otkazana, a ona je stigla na odredište

Napadi na mehanizme za autentifikaciju

- napadi sa preplitanjem (*interleaving attacks*)
  - napadač ubacuje lažne poruke u sklopu protokola da bi ga poremetio

  - primer protokola:
    - korisnik→sistem: c(*k*, *r*) (korisnik šalje sistemu šifrovani slučajan broj, nonce)
    - sistem→korisnik: [*r*, c(*k*, *r'*)] (sistem odgovara dešifrovanim brojem i šalje šifrat novog)
    - korisnik→sistem: *r'* (korisnik odgovara dešifrovanim novim brojem)

  - primeri napada
    - oracle session attack
      - napadač koristi sistem ili klijenta kao dešifrujući orakl
      - cilj: izvući vrednost tajnog broja (nonce) i autentifikacija je uspešna
    - parallel session attack
      - napadač istovremeno vodi više paralelnih sesija sa serverom i korisnikom
      - cilj: preuzeti validne odgovore iz jedne sesije i proslediti ih u drugu

Napadi na mehanizme za autentifikaciju

- *oracle session attack*
  1. korisnik A generiše nonce i šalje sistemu $c_1 = c(k, r_1)$
  2. napadač H presreće poruku c_1
  3. napadač H šalje sistemu $c_1' = c_1$ na dešifrovanje
  4. sistem dešifruje c_1' i vraća $[r_1, c_2(k, r_2)]$
  5. napadač H presreće odgovor i beleži r_1 (tako zna šta se nalazi u c_1)
  6. napadač H šalje sistemu $c_2' = c_2$
  7. sistem dešifruje c_2' i vraća $[r_2, c_3(k, r_3)]$
  8. napadač H šalje sistemu r_2
  9. sistem proverava r_2 i autentifikuje napadača H

- način da se ovaj napad izbegne je da se u koraku 4 poruka u protokolu formira ovako:
  $[c(k, r_1), c(k, r_2)]$
  umesto ovako:
  $[r_1, c(k, r_2)]$

Napadi na mehanizme za autentifikaciju

- *parallel session attack*: napad na modifikovani protokol; odvija se u dve sesije
  - pretpostavlja se da u protokolu obe strane mogu da iniciraju autentifikaciju
  - napadač H presreće komunikaciju između korisnika A i sistema
- napad:
  1. korisnik A šalje inicijalni zahtev sistemu $c_1 = c(k, r_1)$
  2. napadač H presreće c_1
  3. napadač H pokreće paralelnu sesiju ka sistemu sa $c_1' = c_1$
  4. sistem odgovara na paralelnu sesiju napadaču H sa $c_2 = c(k, r_2)$
  5. napadač prosleđuje c_2 korisniku A $c_2' = c_2$
  6. korisnik A dešifruje c_2' da dobije r_2 i formira odgovor $c_3=c(k, f(r_2))$ gde je $f(r_2)$ odgovor na r_2
  7. korisnik A vraća odgovor c_3 u prvoj (originalnoj) sesiji sistemu
  8. napadač H presreće odgovor c_3
  9. napadač H prosleđuje c_3 sistemu u paralelnoj sesiji $c_3' = c_3$
  10. sistem dešifruje c_3' sa $f(r_2)$ i napadač H je uspešno autentifikovan
- protivmere:
  - uključivanje jedinstvenog ID sesije u svaki challenge i odgovor: $c(k, r_1 \,||\, session\_id)$
  - korišćenje timestamp ili sekvence unutar šifrovanja da se otkriju replay poruke

Mere protiv napada na mehanizme za autentifikaciju

- *replay attacks*: koristiti sequence numbers
- *interleaving attacks*: sequence numbers
- *forced delay attacks*: slučajni brojevi i redukovani vremenski prozori

Mere protiv napada na mehanizme za autentifikaciju

- ako se koriste lozinke
  - definisati pravila vezana za dužinu i sadržaj lozinki, rok trajanja, vremenski period kada je dozvoljen pristup
- definisati bezbednosnu politiku koja opisuje uslove pod kojima se korisnički nalozi kreiraju, menjaju i brišu
- sprovoditi strategije za ublažavanje rizika
  - analiza mogućih scenarija napada
  - analizirati moguću štetu kod otkrivanja raznih vrsta tajnih informacija
- za dugo otvorene konekcije sprovoditi autentifikaciju periodično
- ako se autentifikacija koristi zajedno sa proverom integriteta, koristiti različite tajne informacije (ključeve)
- ako se koriste timestamps, neophodna je sinhronizacija časovnika
- ograničiti broj pokušaja za autentifikaciju

Standardi za autentifikaciju

- X.509
- Kerberos
- HTTP basic/digest auth
- OAuth

X.509

- hijerarhijski direktorijumski servis koji čuva informacije o korisnicima za potrebe autentifikacije
- informacije su smeštene u sertifikate potpisane tajnim ključem treće strane kojoj svi veruju
- definiše tri tipa autentifikacione procedure
  - one-way authentication
  - two-way authentication
  - three-way authentication

X.509

- one-way authentication
  - prover šalje verifieru informacije kojima se potvrđuje
    - da je prover formirao poruku
    - da je poruka upućena verifieru
    - da nije bila modifikovana ili ponovo poslata (replay) u prenosu
  - utvrđuje se samo identitet provera

  - poruka mora da sadrži minimalno
    - slučajan broj *rand*, kojim se sprečava *replay* napad
    - timestamp *tmp*, koji uključuje vreme formiranja poruke
    - identitet verifiera
    - potpisani podaci koji uključuju *rand* i *tmp*, praćeni sertifikatom provera
  - a opciono i
    - podaci šifrovani javnim ključem verifiera koji mogu poslužiti za uspostavljanje session ključa

X.509

- two-way authentication
  - omogućava obostranu autentifikaciju
  - dodatna poruka kojom verifier potvrđuje svoj identitet
    - primljeni slučajni broj *rand*
    - novi slučajni broj *rand'*
    - potpisani podaci koji uključuju *rand*, *rand'* i identitet provera kojima se potvrđuje identitet verifiera

X.509

- three-way authentication
  - uključuje i treću poruku koju prover šalje verifieru, koja sadrži
    - slučajan broj *rand'*
    - potpisane podatke koji služe za sinhronizaciju časovnika

X.509

- one-way i two-way autentifikacija se koriste na webu u okviru SSL protokola
- biće reči o tome kasnije

Kerberos

- nastao na MIT-u 1980-tih
- verzije 1-3 su se koristile samo interno na MIT-u
- verzija 4 u javnoj upotrebi
- verzija 5 ispravlja neke nedostatke i postaje široko rasprostranjena (Windows)

- Kerberos = centar za distribuciju ključeva ima tri „glave“
  - authentication
  - authorization
  - accounting

- omogućava **single sign-on** (SSO): korisnik se prijavi samo jednom a potom (u skladu sa svojim pravima) ima pristup svim resursima na mreži
  - upravljanje velikim brojem korisničkih naloga
  - efikasan pristup pojedinačnim resursima
  - lozinke se nikad ne šalju kao otvoreni tekst

Kerberos

- nastao na MIT-u 1980-tih
- verzije 1-3 su se koristile samo interno na MIT-u
- verzija 4 u javnoj upotrebi
- verzija 5 ispravlja neke nedostatke i postaje široko rasprostranjena (Windows)

- Kerberos = centar za distribuciju ključeva ima tri „glave“
  - klijent
  - server
  - Key Distribution Center (KDC)

- omogućava **single sign-on** (SSO): korisnik se prijavi samo jednom a potom (u skladu sa svojim pravima) ima pristup svim resursima na mreži
  - upravljanje velikim brojem korisničkih naloga
  - efikasan pristup pojedinačnim resursima
  - lozinke se nikad ne šalju kao otvoreni tekst

Kerberos

- nastao na MIT-u 1980-tih
- verzije 1-3 su se koristile samo interno na MIT-u
- verzija 4 u javnoj upotrebi
- verzija 5 ispravlja neke nedostatke i postaje široko rasprostranjena (Windows)

- Kerberos = centar za distribuciju ključeva ima tri „glave“
  - bazu podataka
  - server za proveru identiteta
  - server za izdavanje karata

- omogućava **single sign-on** (SSO): korisnik se prijavi samo jednom a potom (u skladu sa svojim pravima) ima pristup svim resursima na mreži
  - upravljanje velikim brojem korisničkih naloga
  - efikasan pristup pojedinačnim resursima
  - lozinke se nikad ne šalju kao otvoreni tekst

Kerberos

- *Aside from the 3-headed dog guarding Hades, the name given to the Athena authentication service, the protocol used by that service, and the libraries used to invoke the authentication and authorization services.*[1]

[1]https://ftp.cs.toronto.edu/doc/athena/kerberos/techplan.txt

Kerberos

- centar za proveru identiteta → Key Distribution Center, KDC
  - baza u kojoj se čuvaju provereni parametri svih učesnika Kerberos sistema
  - svi učesnici mu bezuslovno veruju
  - svaki učesnik (korisnik, računar, mrežni servis) predstavljen je imenom u KDC bazi

Kerberos principal

- koncept principala: jednoznačno identifikuje učesnika
  - za principala je vezan tajni ključ za autentifikaciju učesnika kod KDC-a
- struktura principala: identity/instance@realm
  - *identity*: ime Kerberos učesnika; obavezno polje
  - *instance*: ime računara na kome se nalazi resurs, ili ime korisničke grupe; nije obavezno
  - *realm*: identifikator pojedinačnih Kerberos okruženja; po konvenciji formira se kao uppercase DNS ime domena organizacije, npr. FTN.UNS.AC.RS

- primeri principala:

goran/nastavnici@FTN.UNS.AC.RS	(korisnik goran član grupe nastavnici)
ssh/informatika.ftn.uns.ac.rs@FTN.UNS.AC.RS	(ssh servis na računaru informatika.ftn.uns.ac.rs)
zozon/zozon.ftn.uns.ac.rs@FTN.UNS.AC.RS	(računar zozon.ftn.uns.ac.rs)

Kerberos KDC

- sastoji se iz tri komponente
  - baze principala: evidencija svih principala i njihovih tajnih ključeva
    - implementacija baze: na Linuxu LDAP, na Windowsu Active Directory
  - server za autentifikaciju (*authentication server*, AS)
    - izdaje TGT kartu svim klijentima prilikom prijave na sistem
    - pomoću nje klijenti kasnije traže pristup resursima
    - TGT = ticket-granting ticket
  - server za dodelu karata (*ticket granting server*, TGS)
    - zadužen za izdavanje ST karata namenjenih pojedinim resursima
    - pomoću ST karata klijenti pristupaju resursima
    - ST = service ticket

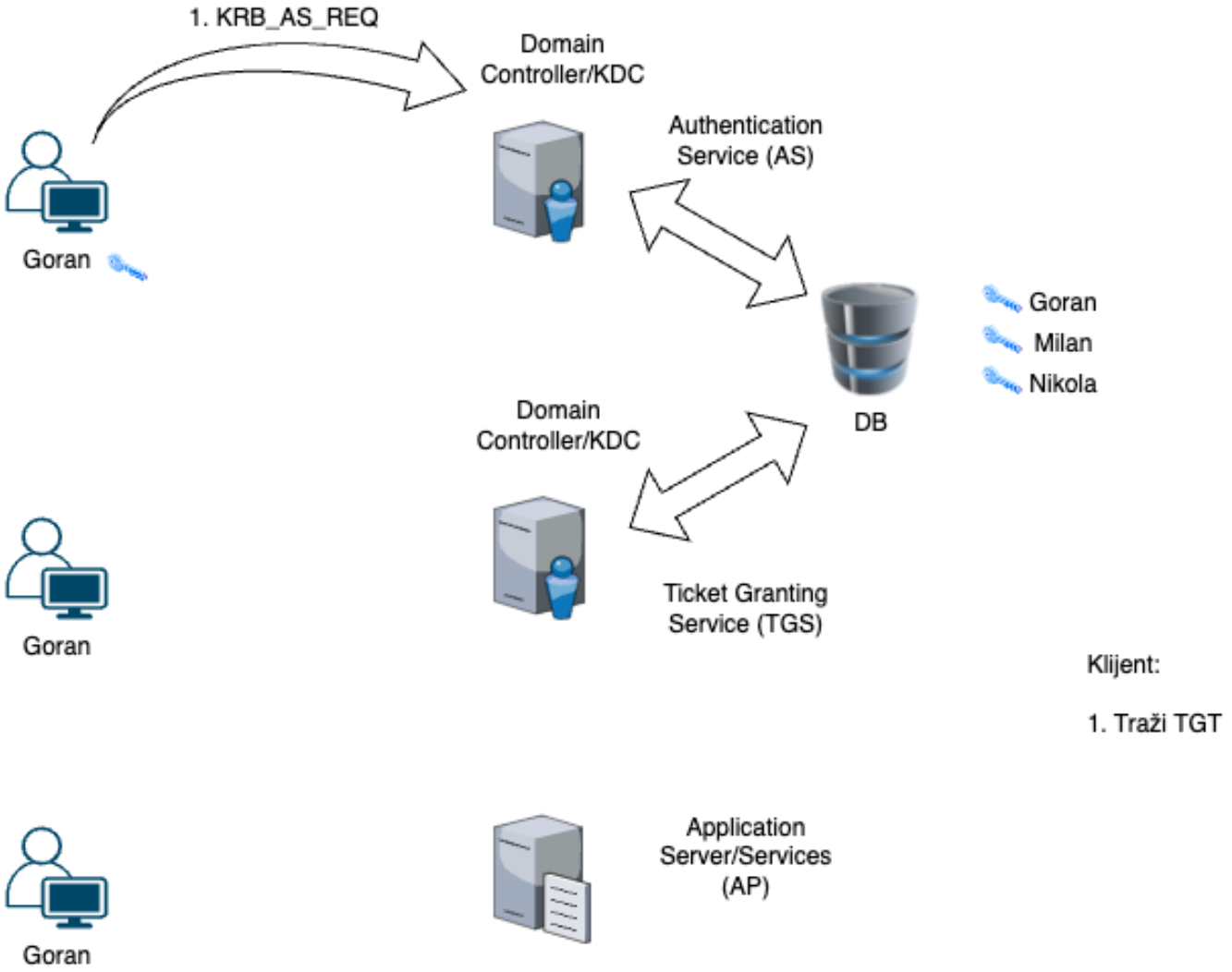
Kerberos KDC

- KDC mora da funkcioniše da bi klijenti mogli pristupati resursima
- potencijalno usko grlo i SPoF (single point of failure)
  - KDC na više servera
    - 1 master KDC
    - više slave KDC - preuzimaju ulogu ukoliko je primarni nedostupan
    - izmene baze samo na primarnom serveru!

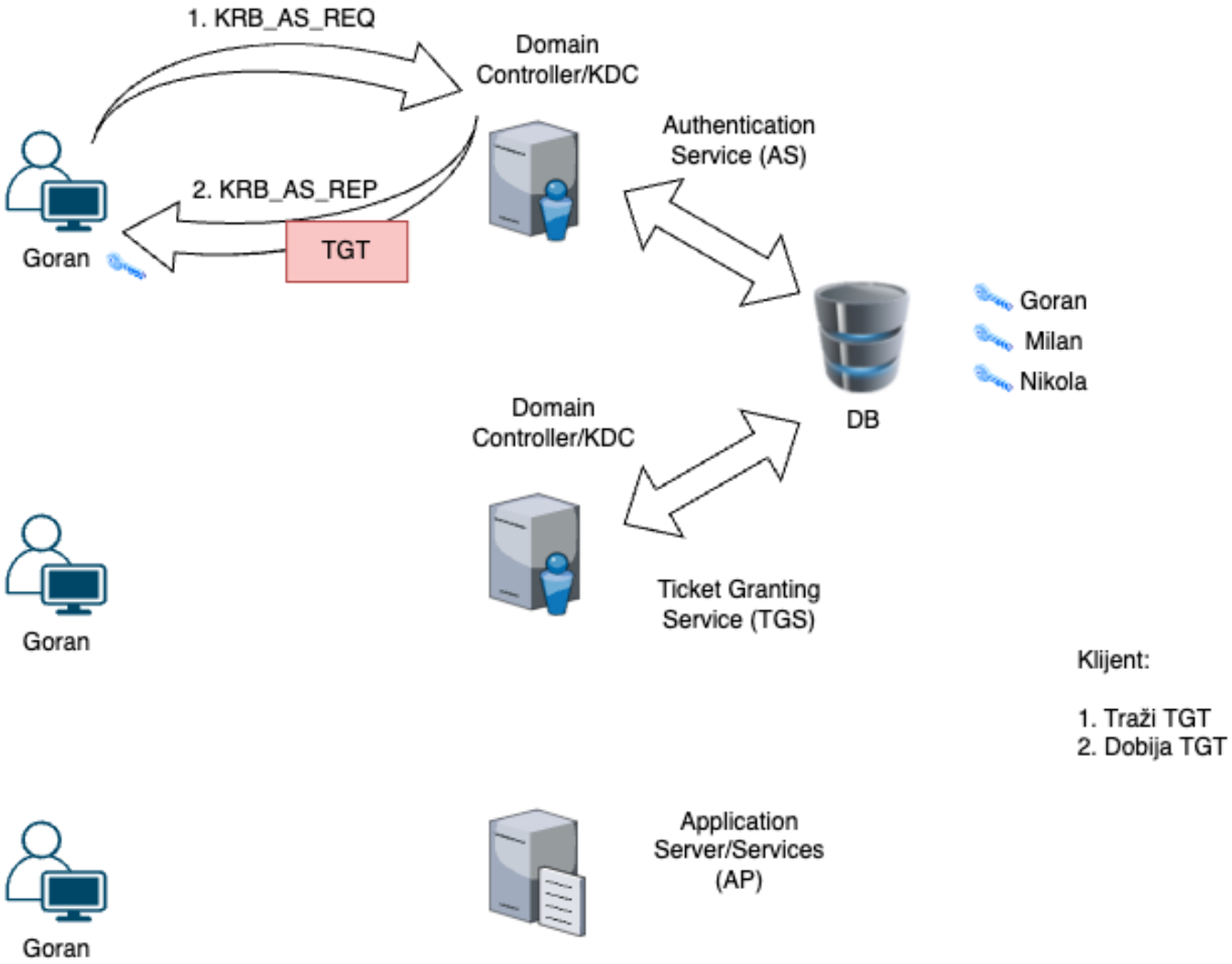
Ticket-granting ticket

- TGT karta sadrži
  - ime principala koji traži pristup
  - ime principala kome se želi pristupiti
  - timestamp (trenutak formiranja zahteva)
  - rok važenja karte
  - listu IP adresa sa kojih se može koristiti karta
  - tajni ključ za komunikaciju sa resursom

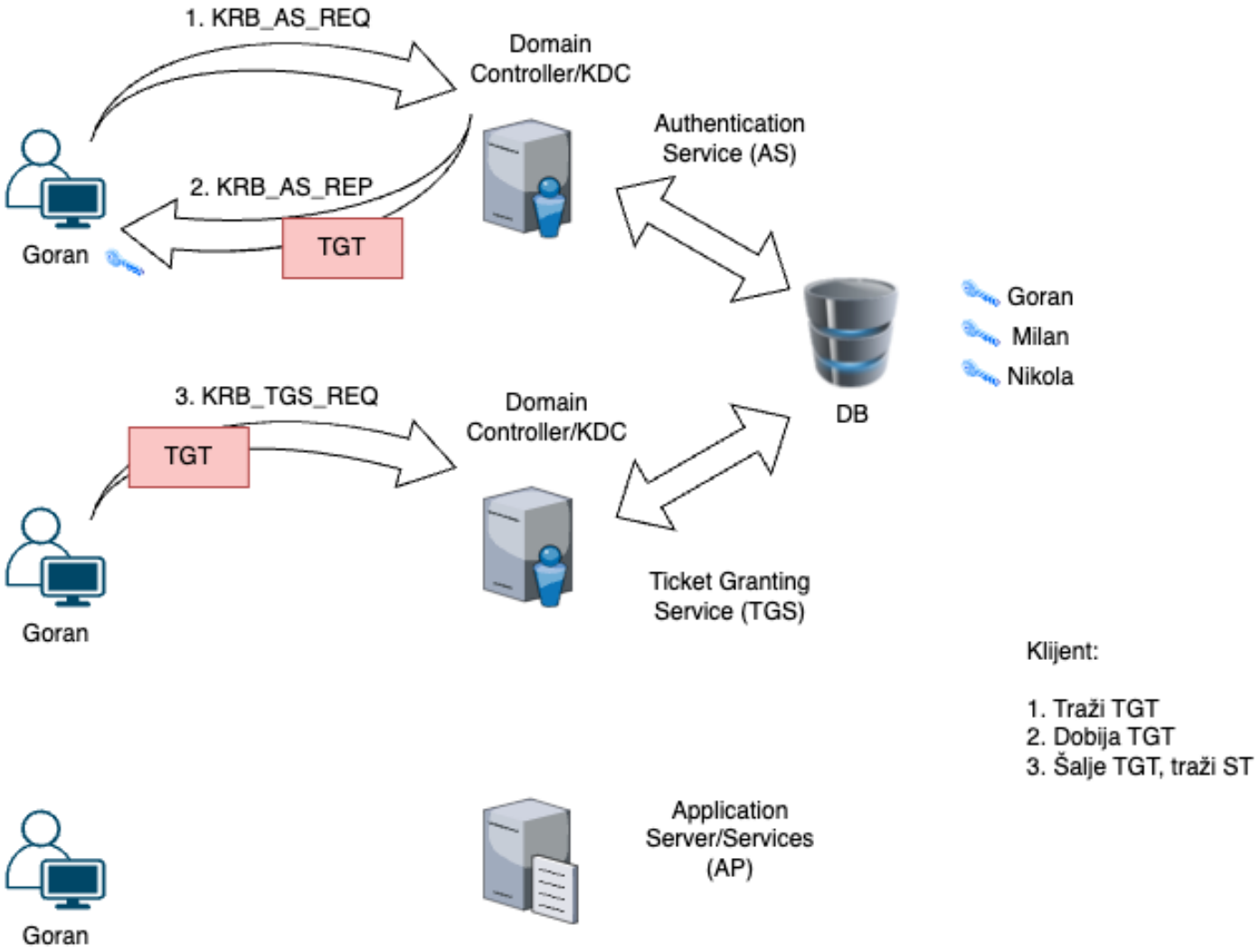
Kerberos - tok komunikacije



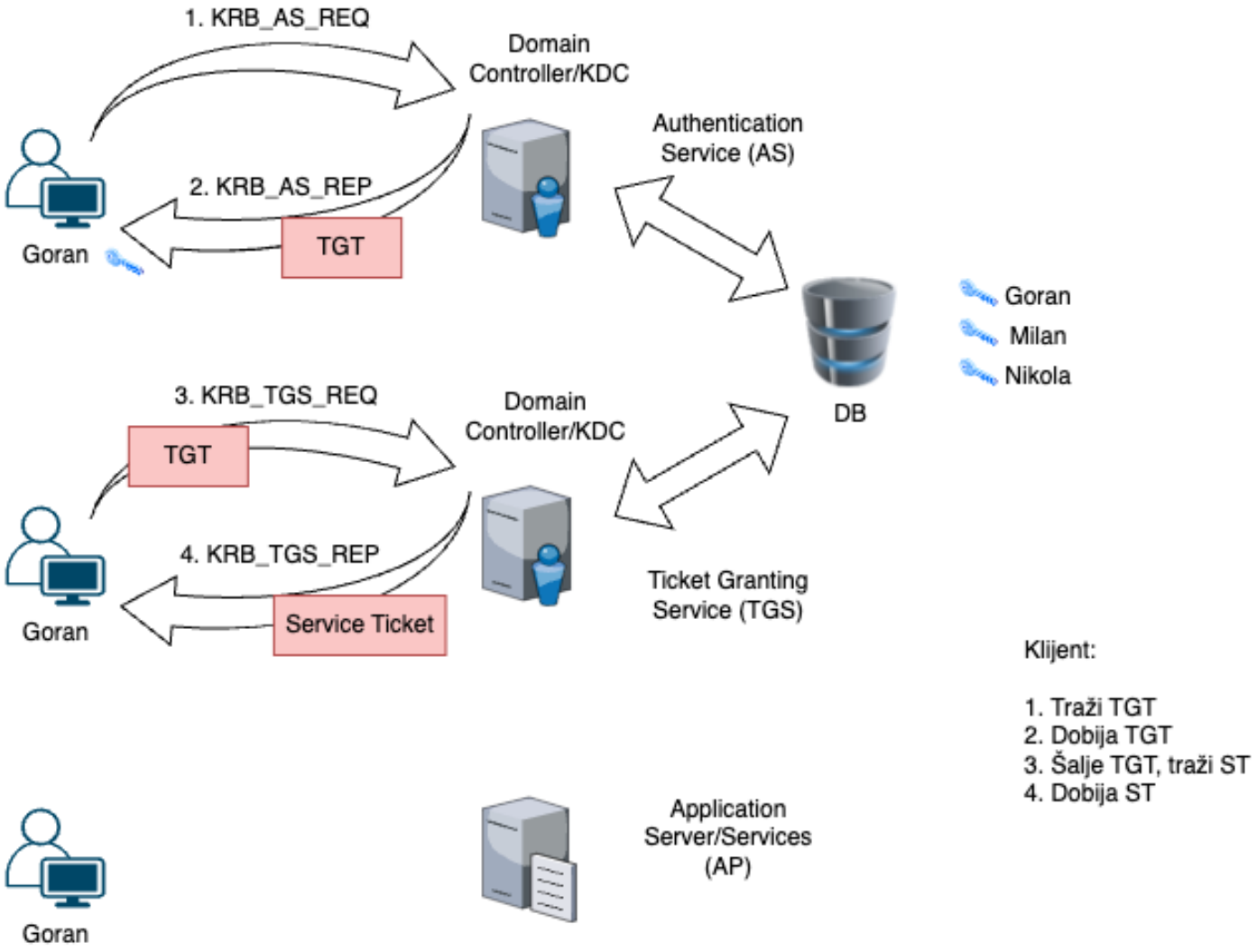
Kerberos - tok komunikacije



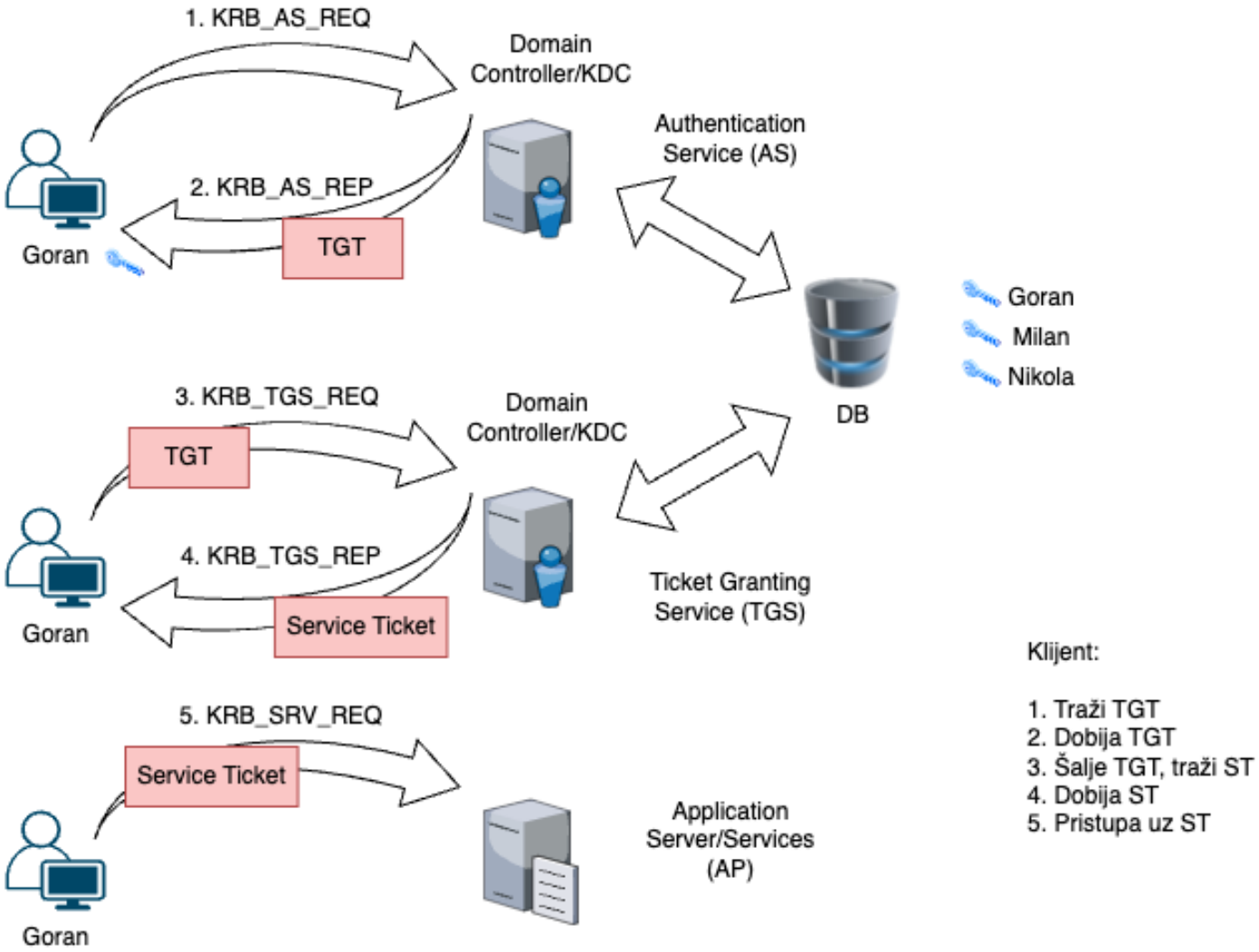
Kerberos - tok komunikacije



Kerberos - tok komunikacije



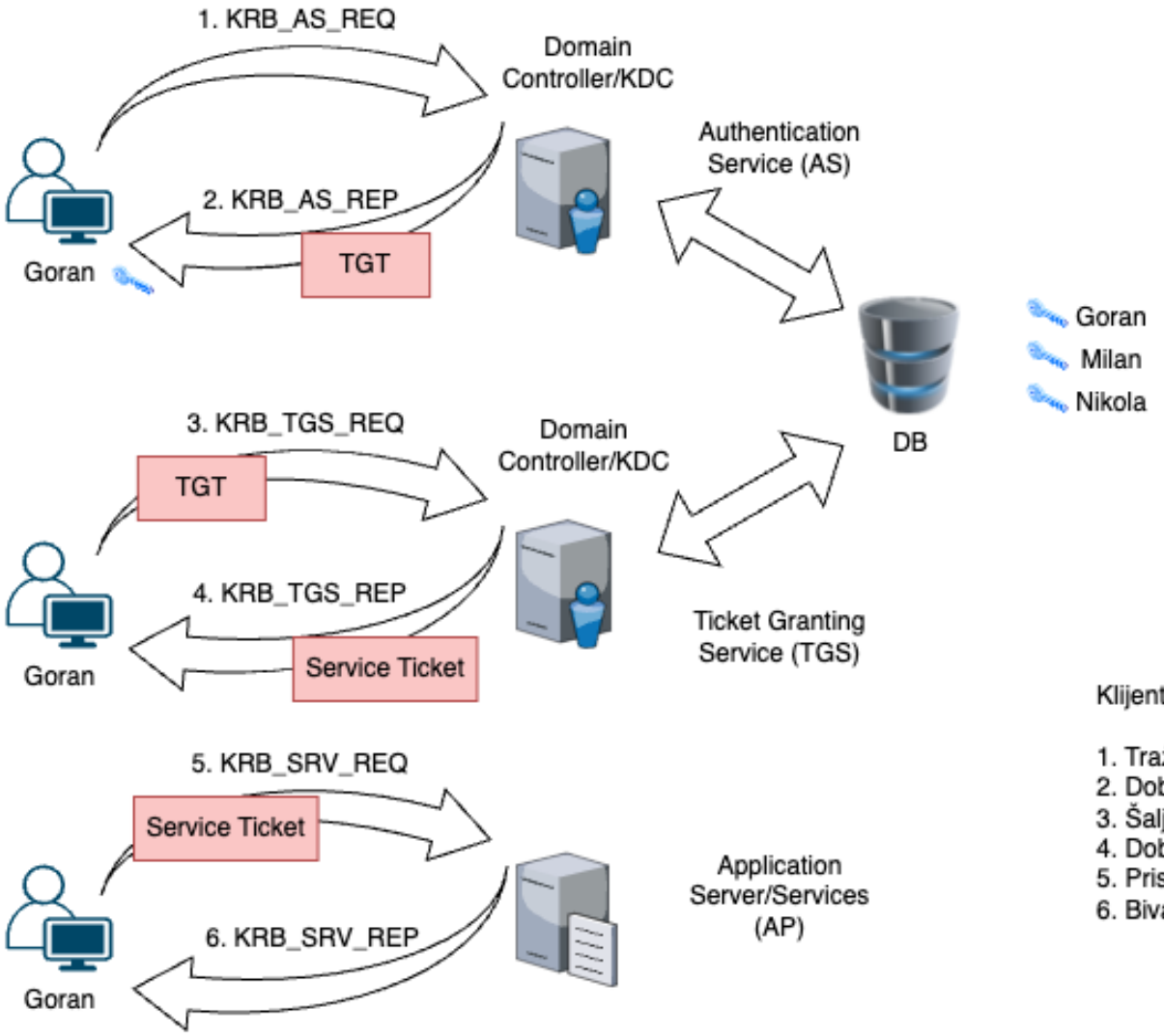
Kerberos - tok komunikacije



Klijent:

1. Traži TGT
2. Dobija TGT
3. Šalje TGT, traži ST
4. Dobija ST
5. Pristupa uz ST

Kerberos - tok komunikacije



Klijent:

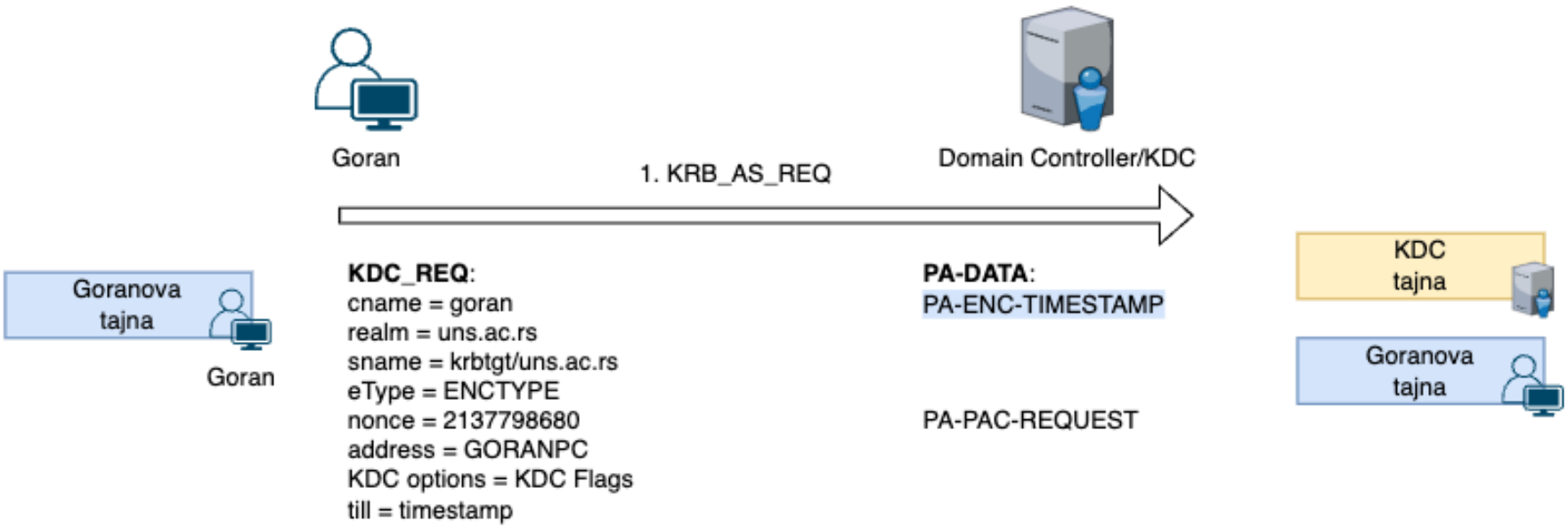
1. Traži TGT
2. Dobija TGT
3. Šalje TGT, traži ST
4. Dobija ST
5. Pristupa uz ST
6. Biva uslužen

Kerberos - tok komunikacije

1. KRB_AS_REQ zahtev - počinje autentifikaciju na KDC AS serveru

- šalje se kao otvoreni tekst
- sadrži
  - ime principala koji šalje zahtev
  - timestamp - vreme generisanja zahteva
  - ime principala TGS servera
  - traženi rok važenja karte

Kerberos - tok komunikacije

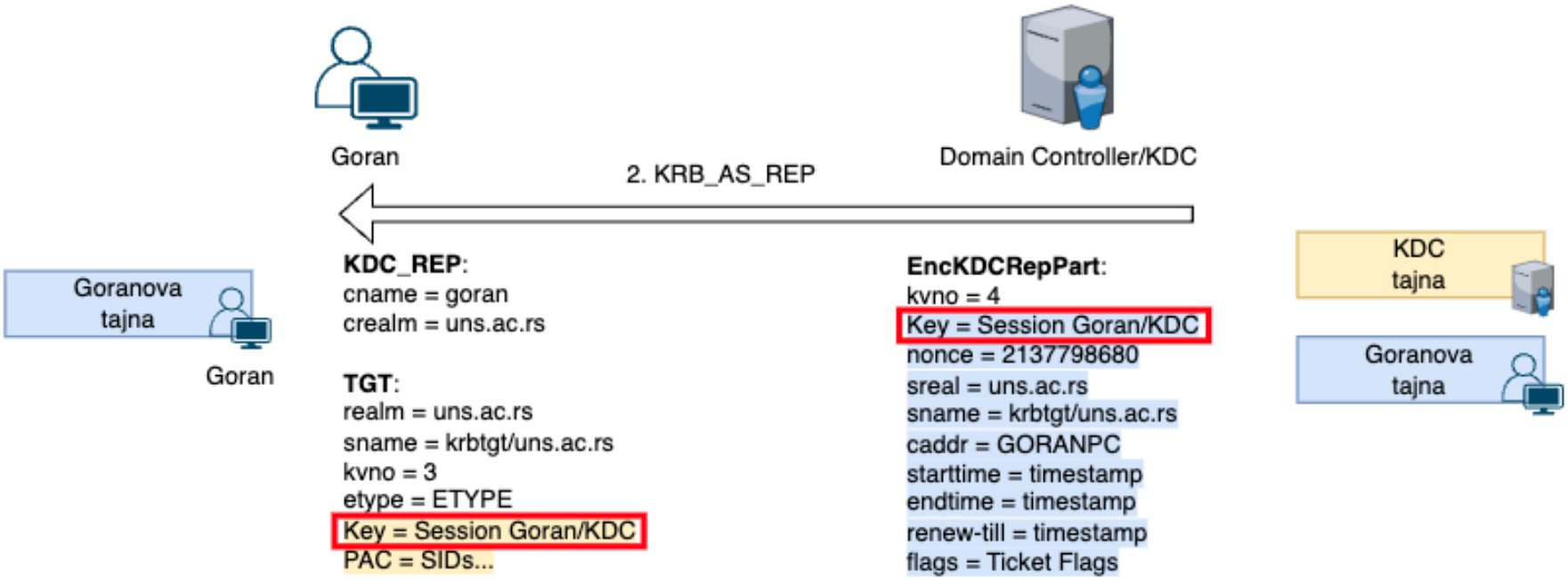


Kerberos - tok komunikacije

2. KRB_AS_REP odgovor

- AS proverava da li principal postoji u bazi
- ako postoji, proverava se da li je timestamp u dozvoljenom vremenskom okviru
- odgovor ima dva dela
  - prvi deo se šifruje tajnim ključem poznatom samo KDC-u i učesniku
- prvi deo sadrži
  - ključ sesije koji će klijent u nastavku koristiti za komunikaciju sa TGS serverom
  - ime principala TGS servera
  - rok važenja karte
- drugi deo sadrži
  - TGT kartu šifrovanu tajnim ključem koji koriste AS i TGS (TGS key)
    - session ključ za komunikaciju klijent-TGS
    - ime principala Kerberos klijenta
    - rok važenja karte
    - timestamp KDC servera
    - klijentova IP adresa
  - klijent ne može da pročita ovaj deo poruke!
  - klijent će je sačuvati u kešu i koristiti prilikom slanja zahteva za pristup resursima

Kerberos - tok komunikacije

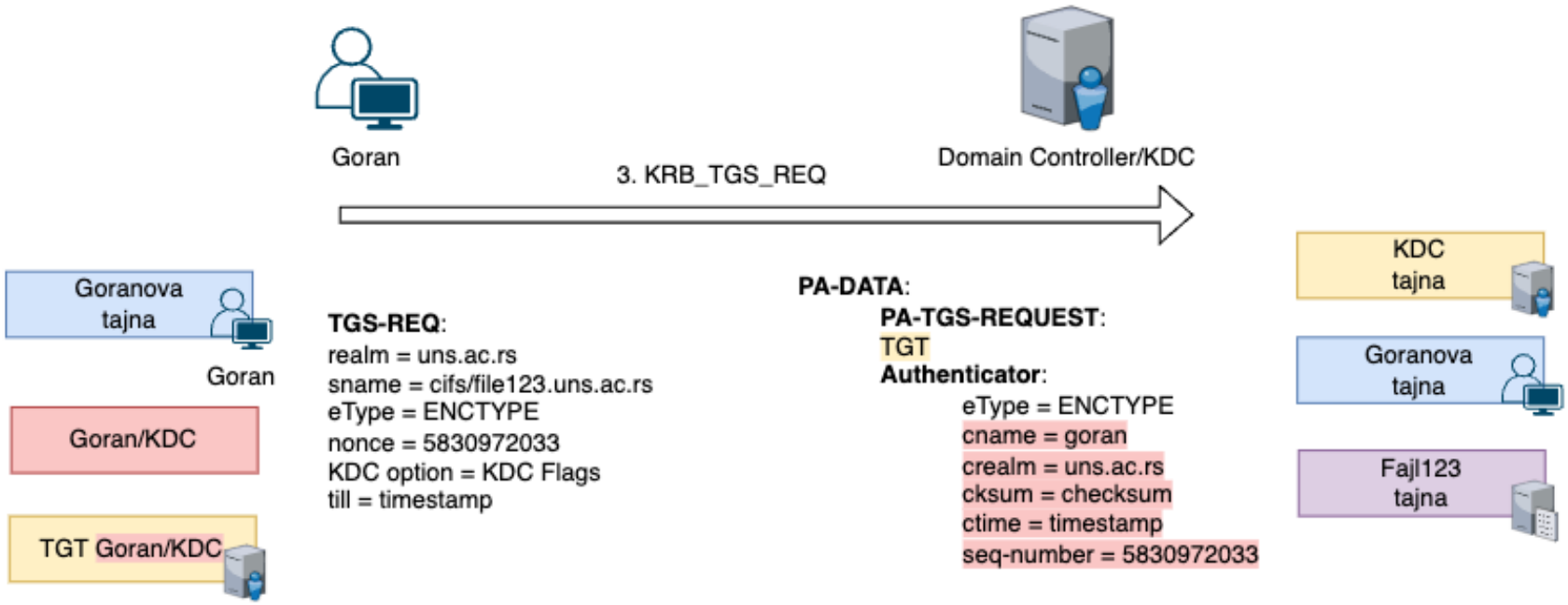


Kerberos - tok komunikacije

3. KRB_TGS_REQ zahtev

- klijent dešifruje prvi deo KRB_AS_REP odgovora
- ključ sesije i šifrovanu TGT kartu smešta u keš
- zahtev za pristup konkretnom resursu klijent šalje TGS serveru
- sadrži
  - ime principala resursa kome klijent želi da pristupi
  - traženi rok važenja karte
  - TGT kartu iz prethodnog koraka
  - autentifikator
    - obezbeđuje jedinstvenost svakog zahteva za pristup resursu
    - potvrđuje da korisnik ima prethodno ugovoren tajni ključ sesije

Kerberos - tok komunikacije

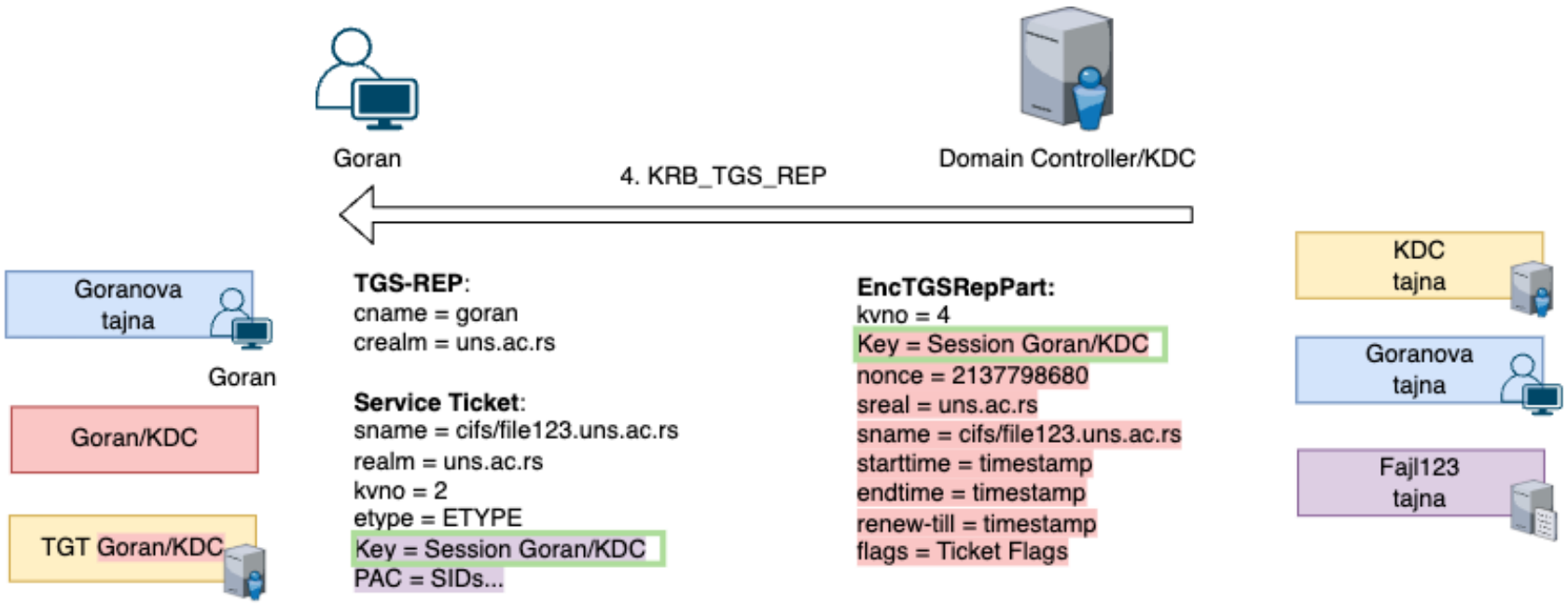


Kerberos - tok komunikacije

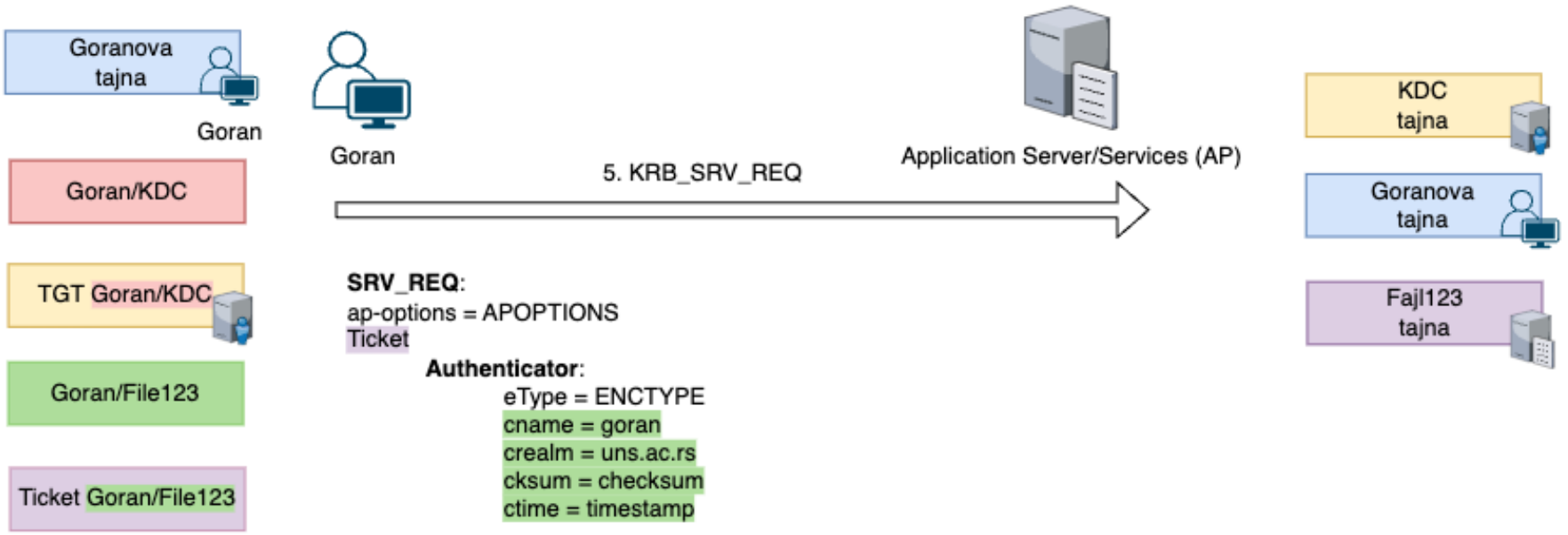
- KRB_TGS_REP odgovor
- TGS server formira novi ključ sesije koji će klijent koristiti za komunikaciju sa resursom
- odgovor (ponovo) ima dva dela
- prvi deo je šifrovan ključem sesije koji je ustanovljen u koracima 1 i 2
  - ime principala resursa kome klijent želi da pristupi
  - rok važenja karte
  - ključ sesije za komunikaciju sa resursom kome klijent želi da pristupi
- drugi deo je ST karta za pristup resursu šifrovana tajnim ključem koji dele TGS i resurs
  - ključ sesije za komunikaciju klijent-resurs
  - ime principala klijenta
  - rok važenja karte
  - timestamp KDC servera
  - klijentova IP adresa

- ... klijent dalje pristupa resursu koristeći ST kartu

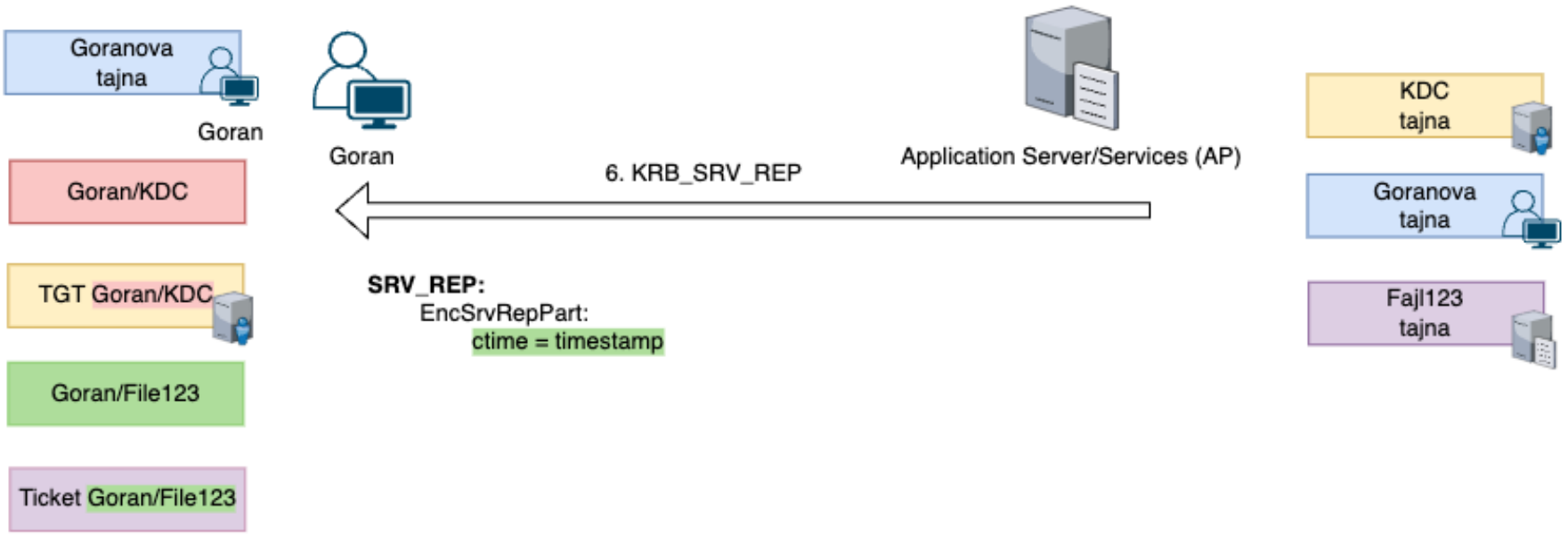
Kerberos - tok komunikacije



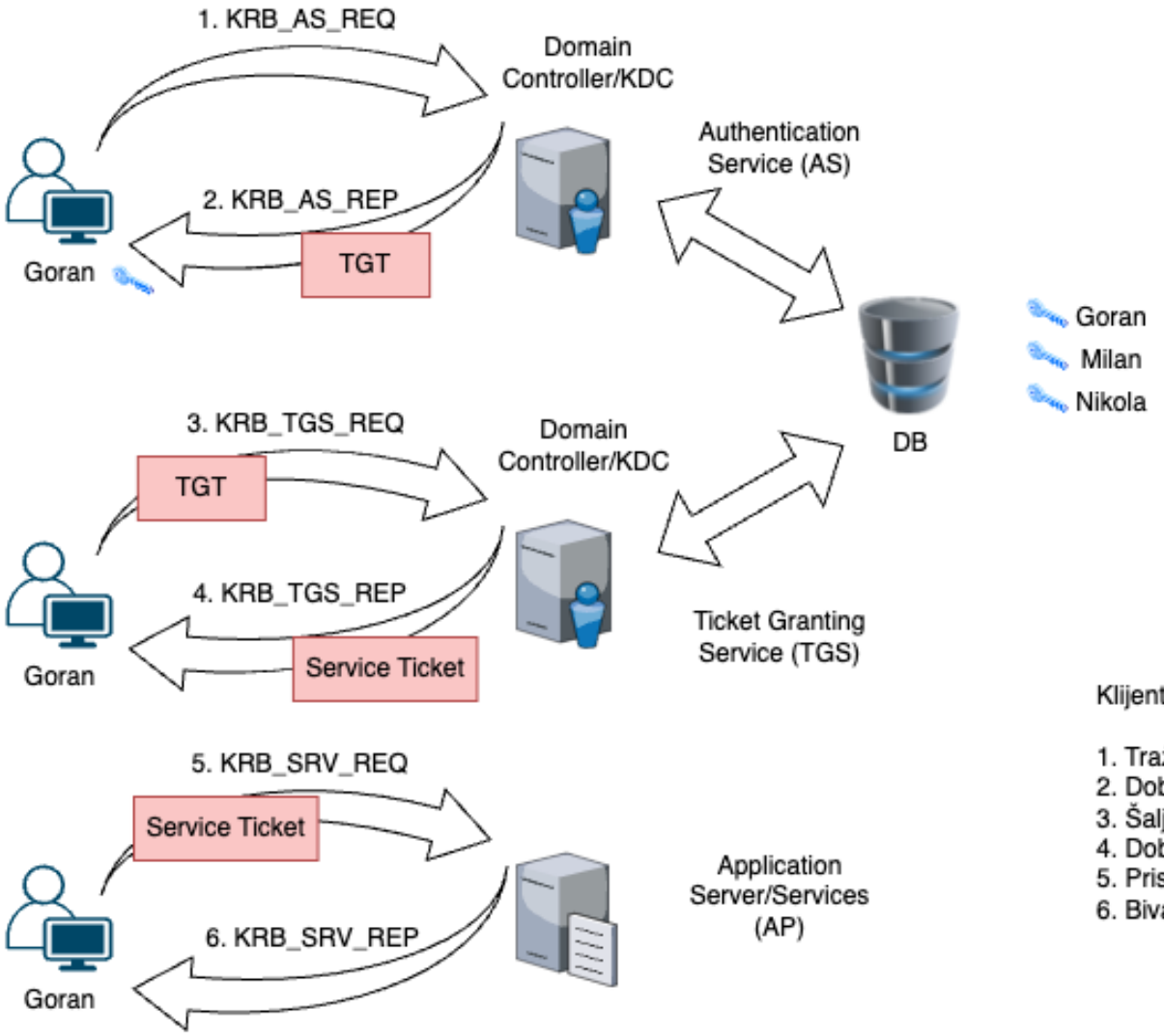
Kerberos - tok komunikacije



Kerberos - tok komunikacije



Kerberos - tok komunikacije



Klijent:

1. Traži TGT
2. Dobija TGT
3. Šalje TGT, traži ST
4. Dobija ST
5. Pristupa uz ST
6. Biva uslužen

Kerberos – tipovi ticketa

- Renewable Ticket
  - svaki ticket ima ograničeni rok važenja, van toga ne može se izvršiti razmena za autentifikaciju
  - aplikacije možda žele da zadrže ticket koji može da važi duži vremenski period
  - ovo može da izloži njihov tajni session ključ krađi, a ti ukradeni ključevi bi važili do isteka ticketa
  - korišćenje kratkotrajnih ticketa i dobijanje novih zahtevalo bi od klijenta dugoročan pristup svom tajnom ključu što je još veći rizik
  - klijent koji ima renewable ticket mora da je pošalje, uz ponovnu autentifikaciju u KDC na obnavljanje pre nego što prođe vreme trajanja ticketa (radi se osvežavanje ključa sesije)

Kerberos – tipovi ticketa

- Post Dated Ticket
  - aplikacije mogu ponekad da dobiju ticket za kasnije korišćenje
  - npr. sistem za obradu podataka u batch-u zahteva da ticket bude validan u trenutku kada se batch posao bude obavljao
  - opasno je držati važeće tickete u batch nizu, jer će oni duže biti dostupni podložnije krađi
  - Post Dated ticketi omogućavaju njihovo dobavljanje od AS-a u vreme podnošenja zahteva za batch izvršavanje, ali se oni ostavljaju „u mirovanju“ dok se ne aktiviraju i validiraju daljim zahtevom AS-a

Kerberos – tipovi ticketa

- Proxiable Ticket
  - može biti neophodno da principal dozvoli servisu da izvrši operaciju u njegovo ime
  - servis mora biti u mogućnosti da preuzme identitet klijenta, ali samo za određenu svrhu
  - principal može dozvoliti servisu da preuzme identitet za određenu svrhu tako što će mu dodeliti proxy
  - ovo omogućava klijentu da prosledi proxy serveru da izvrši zahtev u njegovo ime
  - npr. klijent može da da serveru za štampanje proxy pristup svojim datotekama na fajl serveru kako bi zadovoljio zahtev za štampanje

Kerberos – tipovi ticketa

- Forwardable Ticket
  - prosleđivanje autentifikacije je primer proxy-ja u kojem se servisu dodeljuje kompletna upotreba identiteta klijenta
  - npr. korisnik se prijavi na udaljeni sistem i želi da autentifikacija radi sa tog sistema kao da korisnik radi u lokalu

Kerberos – bezbednost

- korisnikov TGT zahtev je slaba tačka protokola
  - koristi se korisnikova tajna za šifrovanje podataka za autentifikaciju (podložno offline dictionary napadima)
  - može se zaštititi pomoću Flexible Authentication Secure Tunneling (FAST) ili tzv. Kerberos armoringa (dostupno na Windows OS)
    - štiti Kerberos podatke pre autentifikacije za KRB_AS_REQ koristeći LSK (randomly generated logon session key) iz TGT-a kao zajedničku tajnu za potpuno šifrovanje Kerberos poruka i potpisivanje svih mogućih Kerberos poruka o greškama
    - zajednička tajna daje dodatni salt u Kerberos procesu autentifikacije
    - ovo rezultuje produženim vremenom obrade, ali ne menja veličinu ticketa
    - zajednička tajna pruža DC-ovima mogućnost da vrate greške o Kerberos autentifikaciji, što zauzvrat štiti od spoofing-a, man-in-the-middle i drugih napada

Kerberos – bezbednost

- ako se ticket ukrade, može se koristiti za lažno predstavljanje korisnika
  - Pass-the-ticket
- ako se lozinka servisnog naloga ukrade, može se koristiti za lažno predstavljanje korisnika
  - Silver ticket
- ako se krbtgt hash ukrade, može se koristiti za lažno predstavljanje bilo koga u komunikaciji
  - Golden ticket

HTTP autentifikacija

- postoje dva tipa autentifikacije po HTTP standardu
  - Basic Authentication
    - relativno često korišćena
  - Digest Access Authentication
    - vrlo retko korišćena
    - često nepravilno implementirana

HTTP Basic Authentication

- klijenti se identifikuju na osnovu korisničkog imena i lozinke

- izvodi se na sledeći način:
  - klijent traži željeni resurs
  - server proveri da li je pristup resursu ograničen
  - ako je ograničen, proveri da li je klijent već ranije poslao username:password
  - ako nije, kao odgovor mu se šalje:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm=OBLAST
```

realm – ime zaštićene oblasti

  - klijent prikaže prozor za unos username:password i ponovo traži željeni resurs
  - ako username:password nisu ispravni, dobija se isti odgovor (401)

HTTP Basic Authentication

- username:password se šalju u okviru zaglavlja HTTP zahteva

```
GET .... HTTP/1.1
...
Authorization: Basic cHJvYmE6cHJvYmE=
```

- ime i lozinka se pakuju kao **ime:lozinka** (razdvojeni dvotačkom) i kodiraju **Base64** algoritmom

- Base64 algoritam nije kriptografski algoritam
  - predstavlja brojeve u brojnom sistemu sa osnovom 64
  - najveća osnova brojnog sistema takva da se brojevi mogu predstaviti ASCII karakterima
  - služi za pakovanje binarnih sadržaja u tekstualni format
    - email poruke sa zakačenim binarnim fajlovima

HTTP Basic Authentication

- jednostavan mehanizam za autentifikaciju

- username:password putuju od klijenta do servera kao otvoreni tekst
- nema zaštite od prisluškivanja
- nizak nivo zaštite

HTTP Digest Access Authentication

- ispravlja osnovnu manu Basic metode
  - korisničko ime i lozinka se ne šalju preko mreže, već samo njihov hash kod
  - na serverskoj strani se ne mora čuvati ime i lozinka, već njihov hash kod

- radi po challenge-response principu:
  - server pošalje klijentu kodiranu informaciju
  - klijent vraća korisničko ime, lozinku i kodiranu informaciju

HTTP Digest Access Authentication

- tok komunikacije
  - klijent traži željeni resurs
  - server proverava da li je pristup resursu ograničen
  - ako jeste, proverava da li je klijent već ranije poslao username:password
  - ako nije, kao odgovor mu se šalje:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm=OBLAST, nonce="...",
...
```

  - klijent prikaže prozor za unos username:password i ponovo traži željeni resurs
  - ako username:password nisu ispravni, dobija se isti odgovor (401)

HTTP Digest Access Authentication

- struktura WWW-Authenticate zaglavlja:

```
WWW-Authenticate: Digest
   realm="IME",
   nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
   qop="auth,auth-int",
   opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

- realm – ime zaštićene oblasti
- nonce – jednokratna slučajna vrednost
- qop (quality of protection) – auth (authentication) i/ili auth-int (authentication with integrity protection)
- opaque – string koji bi klijent trebalo da pošalje za svaki naredni zahtev unutar iste oblasti

HTTP Digest Access Authentication

- odgovor klijenta stiže u sledećem zahtevu, u zaglavlju Authorization:

```
Authorization: Digest
  username="proba",
  realm="IME",
  qop="auth",
  algorithm="MD5",
  uri="/protect.html",
  nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
  nc=00000001,
  cnonce="aa671f12a8587e2cffe73519863f9dbb",
  opaque="5ccc069c403ebaf9f0171e9517f40e41",
  response="f90a24d42f7d59fa0496cbbce6aacfe6"
```

- uri – zahtevani resurs
- nc – brojač zahteva za istim nonce poljem (svaki naredni nc mora biti za 1 veći za isti nonce klijenta, sprečava od reply napada)
- cnonce – koristi se za izračunavanje response
- response – sadrži hash kod za korisničko ime i lozinku

HTTP Digest Access Authentication

- izračunavanje response polja

  HA1=MD5(username + ":" + realm + ":" + password)
  HA2=MD5(http_method + ":" + uri)
  response=MD5(HA1 + ":" + nonce + ":" + nc + ":" + cnonce + ":" + qop + ":" + HA2)

- MD5 – standardna kriptografska hash funkcija

HTTP Digest Access Authentication

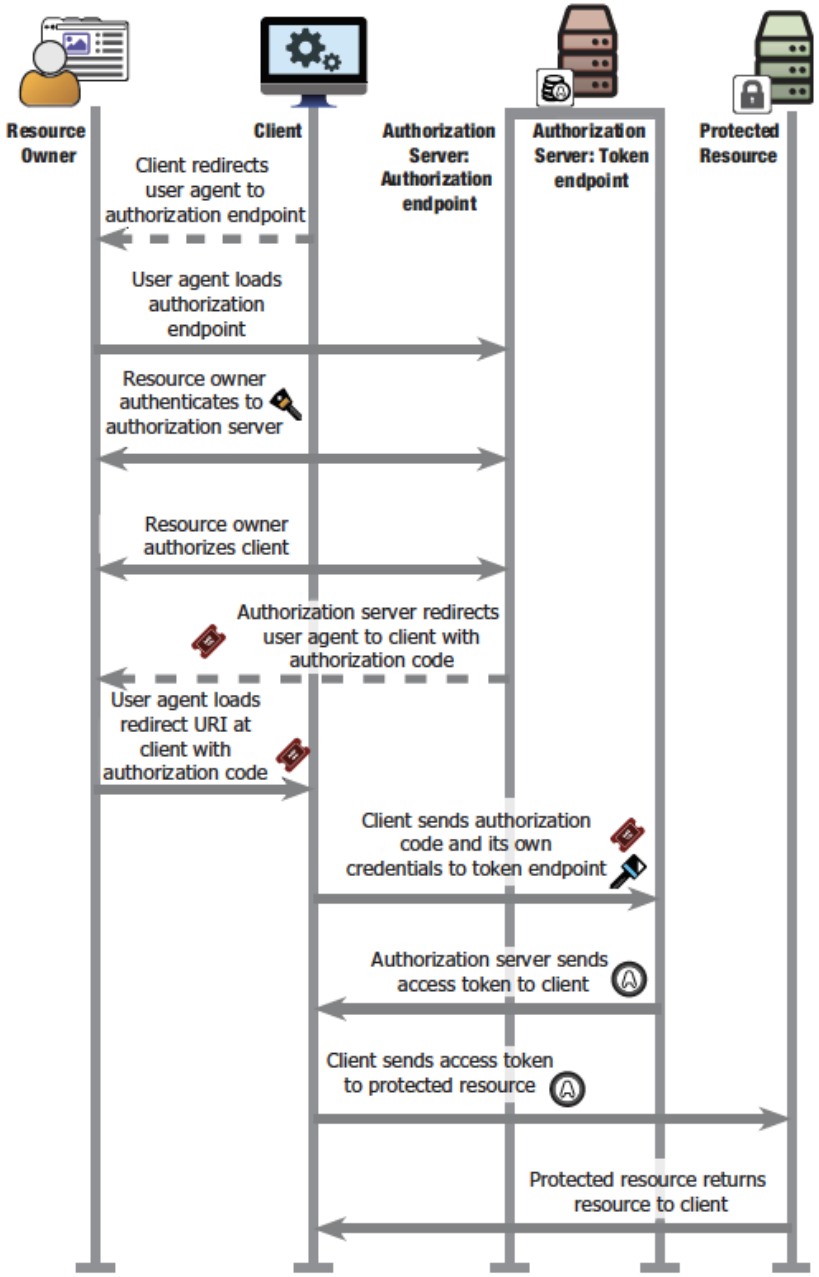
- lozinka ne putuje preko mreže, već samo njen hash
- otkrivanje lozinke prisluškivanjem nije moguće

- preneti sadržaji i dalje nisu zaštićeni od prisluškivanja (samo lozinka jeste)
- nema zaštite od man-in-the-middle napada

OAuth 2.0

- autorizacioni okvir koji omogućuje aplikacijama da pristupaju resursima u ime vlasnika resursa
- protokol za delegaciju
  - neko ko je vlasnik resursa dozvoljava nekom drugom da pristupi resursu u njegovo ime
  - nije baš protokol samo za autentifikaciju
- aplikacija od vlasnika zahtev autorizaciju i dobija token koji koristi za pristup
  - token = delegirano pravo pristupa
- napravljen za web sisteme
  - pre svega REST bazirane
- implicitno uključuje autentifikaciju

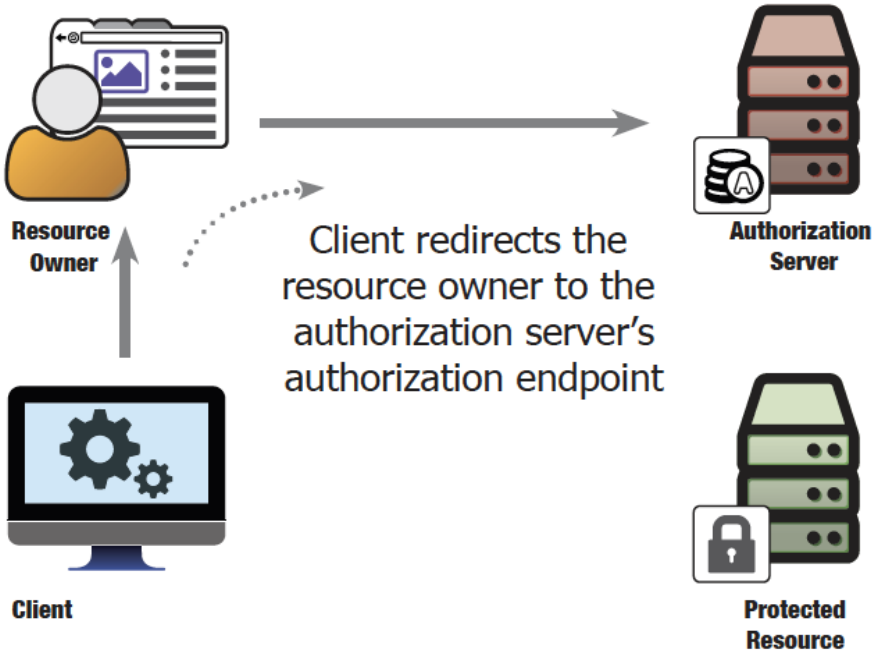
OAuth 2.0


Resource Owner
Client
Authorization Server: Authorization endpoint
Authorization Server: Token endpoint
Protected Resource
Client redirects user agent to authorization endpoint
User agent loads authorization endpoint
Resource owner authenticates to authorization server
Resource owner authorizes client
Authorization server redirects user agent to client with authorization code
User agent loads redirect URI at client with authorization code
Client sends authorization code and its own credentials to token endpoint
Authorization server sends access token to client
Client sends access token to protected resource
Protected resource returns resource to client

OAuth 2.0

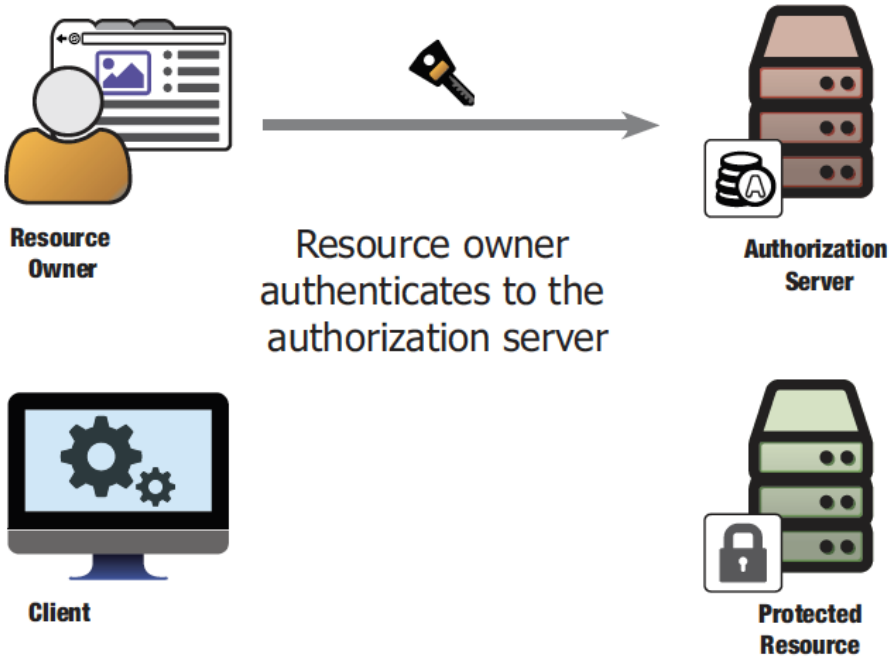
- vlasnik resursa ukazuje klijentskoj aplikaciji da želi da aplikacija uradi operaciju za njega
  - npr. da učita slike sa servisa kako bi se moglo izvršiti štampanje
- kada klijent zaključi da mu treba OAuth access token redirektuje vlasnika resursa na autorizacioni server

```
HTTP/1.1 302 Moved Temporarily
Location: http://localhost:9001/authorize?response_type=code
&scope=foo&client_id=oauthclient1&redirect_uri=http%3A%2F%2Floc
alhost%3A9000%2Fcallback&state=Lwt50DDQKUB8U7jtfLQCVGDL9cnmwHH1
```



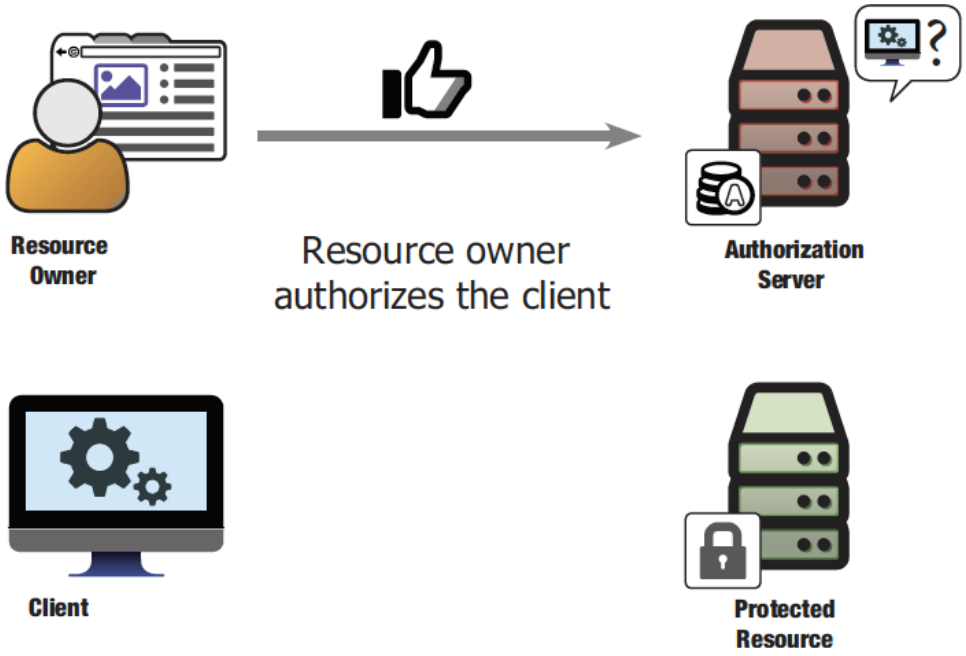
OAuth 2.0

- autorizacioni server zahteva od vlasnika resursa da se autentifikuje
  - ko je vlasnik resursa i šta može da delegira klijentu
  - autentifikacije su uvek između vlasnika resursa i servera
    - klijent ne učestvuje
    - nema deljenja kredencijala sa klijentom
  - način autentifikacije je proizvoljan



OAuth 2.0

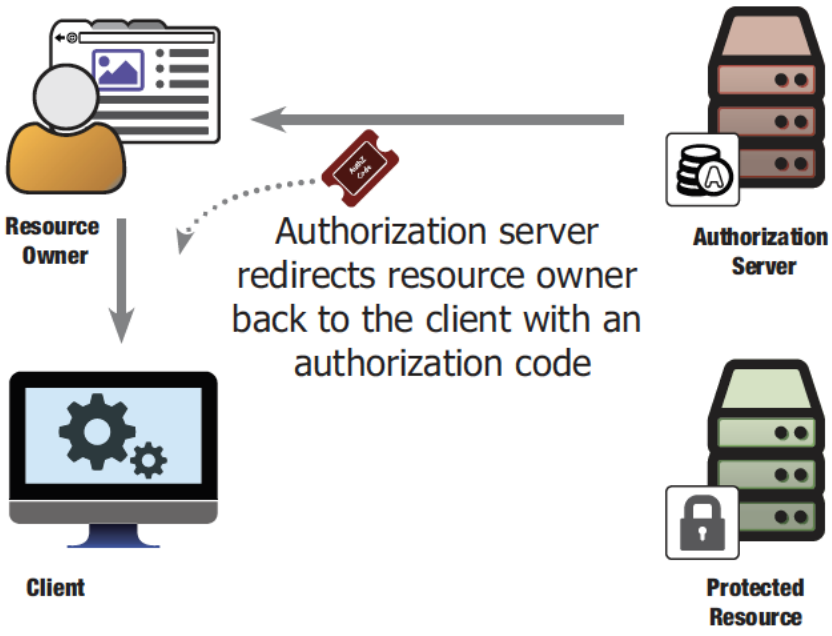
- vlasnik resursa autorizuje klijentsku aplikaciju
  - deo prava se delegira klijentskoj aplikaciji
- server može da kešira odluke za buduće identične zahteve
- server može da „pregazi“ odobrenje vlasnika resursa



OAuth 2.0

- autorizacioni server redirektuje vlasnika resursa nazad na klijentsku aplikaciju
  - u primeru se kao grant_type koristi authorization_code (videćemo sve tipove kasnije)
    - parametar code predstavlja one-time-use kredencijale
      - klijent ga koristi za kasnije aktivnosti
    - parametar state mora biti identičan kao na početku

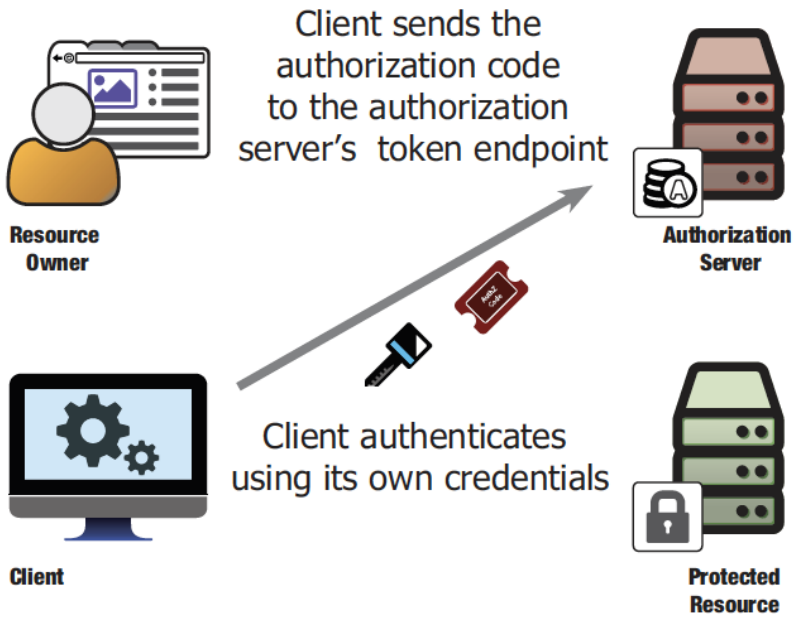
```
HTTP 302 Found
Location: http://localhost:9000/oauth_callback?code=8V1pr0rJ&
state=Lwt50DDQKUB8U7jtfLQCVGDL9cnmwHH1
```



OAuth 2.0

- klijent šalje vrednost parametra code autorizacionom serveru zajedno sa svojim kredencijalima (client_id:client_secret)

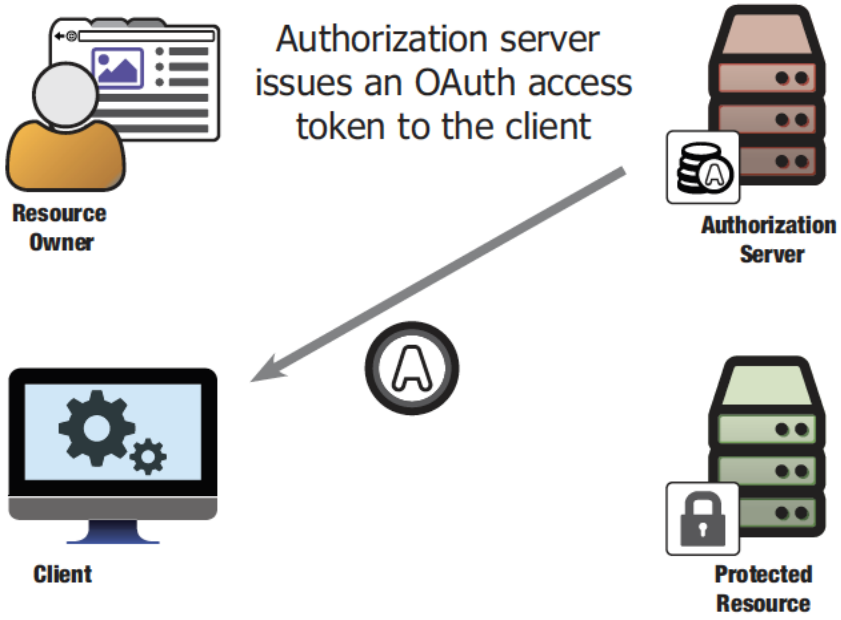
```
POST /token
Host: localhost:9001
Accept: application/json
Content-type: application/x-www-form-urlencoded
Authorization: Basic b2F1dGgtY2xpZW50LTE6b2F1dGgtY2xpZW50LXN
grant_type=authorization_code&redirect_uri=http%3A%2F%2Flocalho
st%3A9000%2Fcallback&code=8V1pr0rJ
```



OAuth 2.0

- autorizacioni server izdaje (access) token klijentu

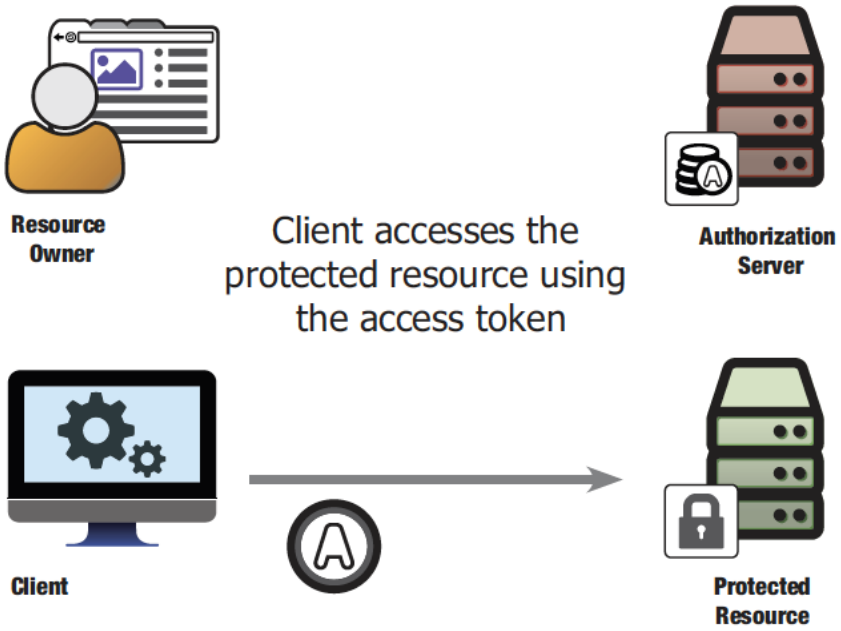
```
HTTP 200 OK
Date: Fri, 31 Jul 2015 21:19:03 GMT
Content-type: application/json
{
    "access_token": "987tghjkiu6trfghjuytrghj",
    "token_type": "Bearer"
}
```



OAuth 2.0

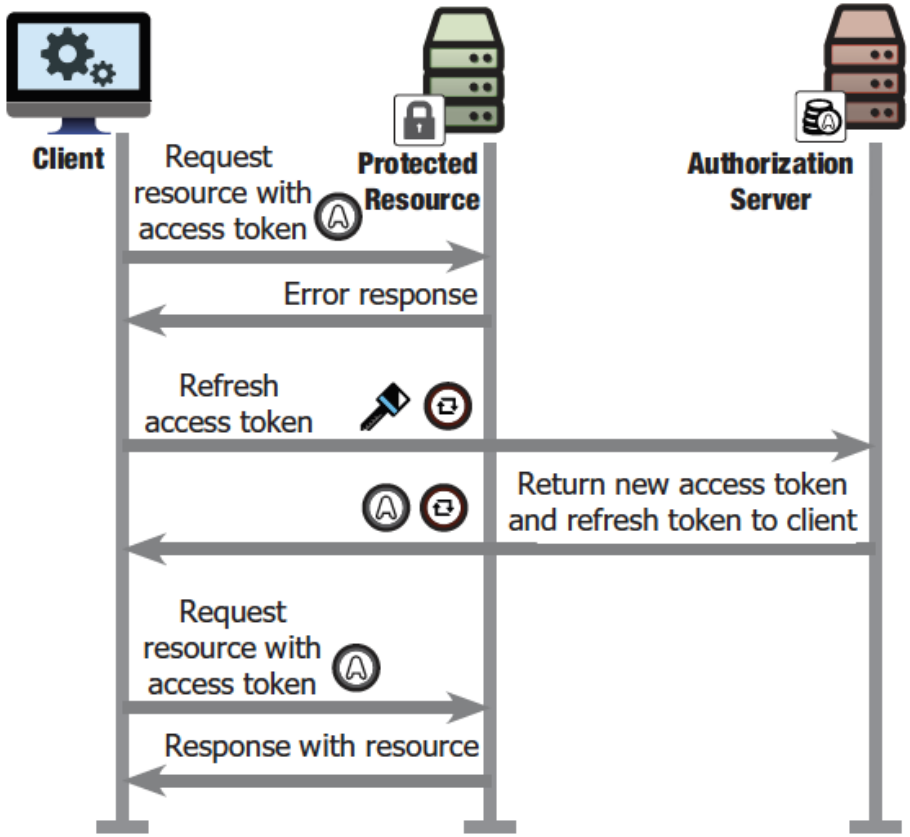
- sa dobijenim (access) tokenom klijent pristupa resursu
  - resurs parsira i validira token

```
GET /resource HTTP/1.1
Host: localhost:9002
Accept: application/json
Connection: keep-alive
Authorization: Bearer 987tghjkiu6trfghjuytrghj
```



OAuth 2.0

- access token može biti povučen ili da ima vreme trajanja
- refresh token
  - služi da klijent zatraži novi access token token bez kontaktiranja resource owner-a



OAuth 2.0

- *authorization token*
  - standard ne definiše format i sadržaj tokena
    - može da sadrži bilo šta
    - obično JSON format
  - klijent nema potrebe za razumevanje tokena
    - samog ga prosleđuje u zahtevu
    - jednostavnija implementacija
  - server mora biti u stanju da pročita i razume token
  - tri načina slanja
    - HTTP authorization header – preporučen način
    - form-encoded request body
    - URL/encoded param

OAuth 2.0

- *scope*
  - skup prava za zaštićeni resurs - grupisanje prava
  - jedan string ili sekvenca stringova razdvojeni razmakom
  - njihovo definisanje obično zavisi od API-ja

OAuth 2.0

- *refresh token*
  - klijent ga koristi za obnavljanje access tokena, kada on istekne bez potrebe da se u obnavljanje uključi korisnik

OAuth 2.0

- vrste klijenata
  - javni
    - ne vrši se autentifikacija klijenta
  - poverljivi
    - klijenti poseduje autentifikacione podatke
    - svaka instanca klijenta ima posebnu konfiguraciju

OAuth 2.0

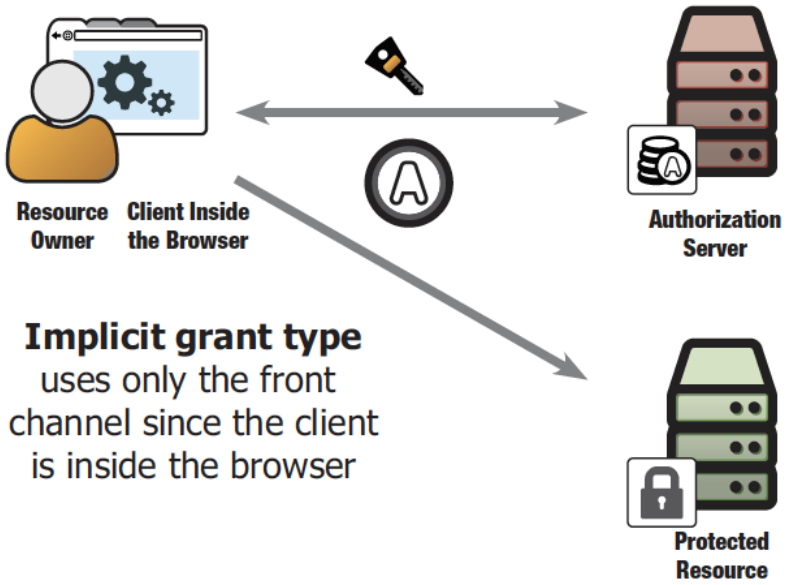
- authorization grant
  - metoda za dobijanje tokena
    - eksplicitni
    - implicitni
    - client credentials grant
    - resource owner credentials grant
    - assertion grant

OAuth 2.0

- eksplicitni authorization grant
  - klijentska aplikacija preusmerava korisnika na autorizacioni server sa response_type=code, client_id, redirect_uri
  - koraci
    1. korisnik se prijavljuje i odobrava pristup
    2. autorizacioni server vraća authorization code (kratkotrajan kod) klijentu kroz redirect_uri
    3. klijent direktno serveru za code (sa tajnim client_secret), uz grant_type=authorization_code, zatraži access token
  - prednosti
    - nikada ne izlaže access_token u browseru
    - client_secret ostaje tajan (čuva se na bekendu)

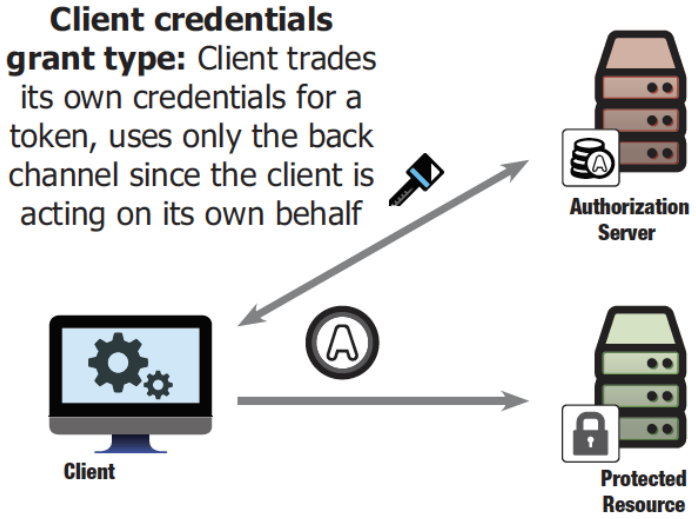
OAuth 2.0

- *implicitni authorization grant*
  - samo klijenti koji ne mogu bezbedno da čuvaju client_secret (SPA ili mobilne aplikacije bez bekenda)
  - koraci
    1. korisnik se preusmerava na autorizacioni server sa response_type=token
    2. nakon odobrenja, autorizacioni server direktno vraća access_token u URI fragmentu (#access_token=…)
  - nedostaci
    - token se izlaže u browseru i rizikuje interception
    - ne postoji refresh token—mora se ponovo proći autorizacija



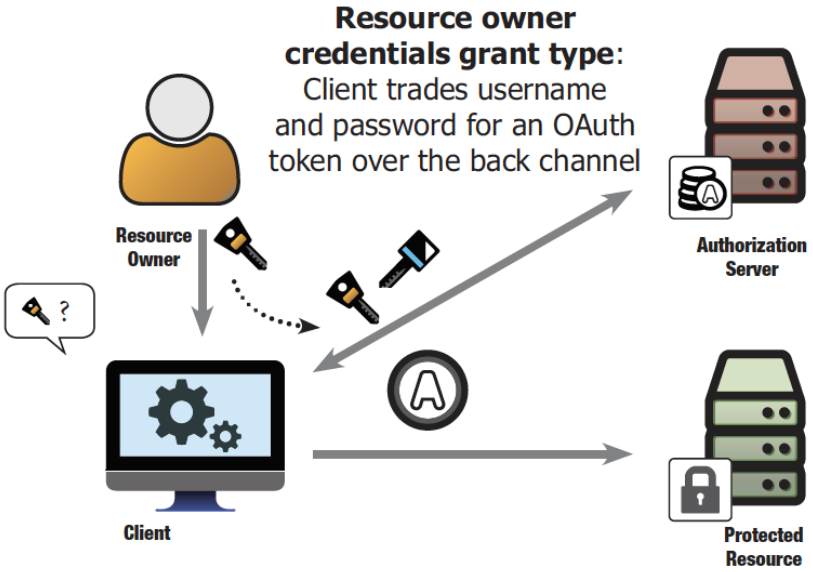
OAuth 2.0

- *client credentials authorization grant*
  - server-server komunikacija, gde klijent sam poseduje privilegiju (nema korisničkog konteksta)
  - koraci
    1. klijent šalje client_id i client_secret direktno autorizacionom serveru (grant_type=client_credentials)
    2. server vraća access_token koji predstavlja samog klijenta (bez korisnika)
  - upotreba
    - servis-to-servis API pozivi
    - automatizovani batch poslovi



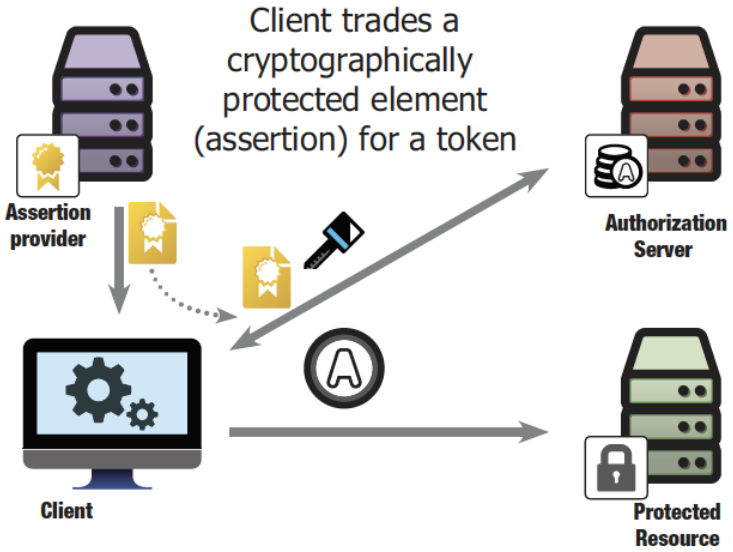
OAuth 2.0

- *resource owner credentials grant*
  - visoko poverljivi klijenti kojima korisnik direktno veruje ili neki CLI alati
  - koraci
    1. korisnik unese username i password u aplikaciju
    2. klijent šalje kredencijale zajedno sa client_id/secret (grant_type=password) na token endpoint
    3. server vraća access_token (i opciono refresh_token)
  - treba biti oprezan
    - klijent direktno vidi korisničku lozinku ⟶ primenjuje se samo kad nema druge opcije



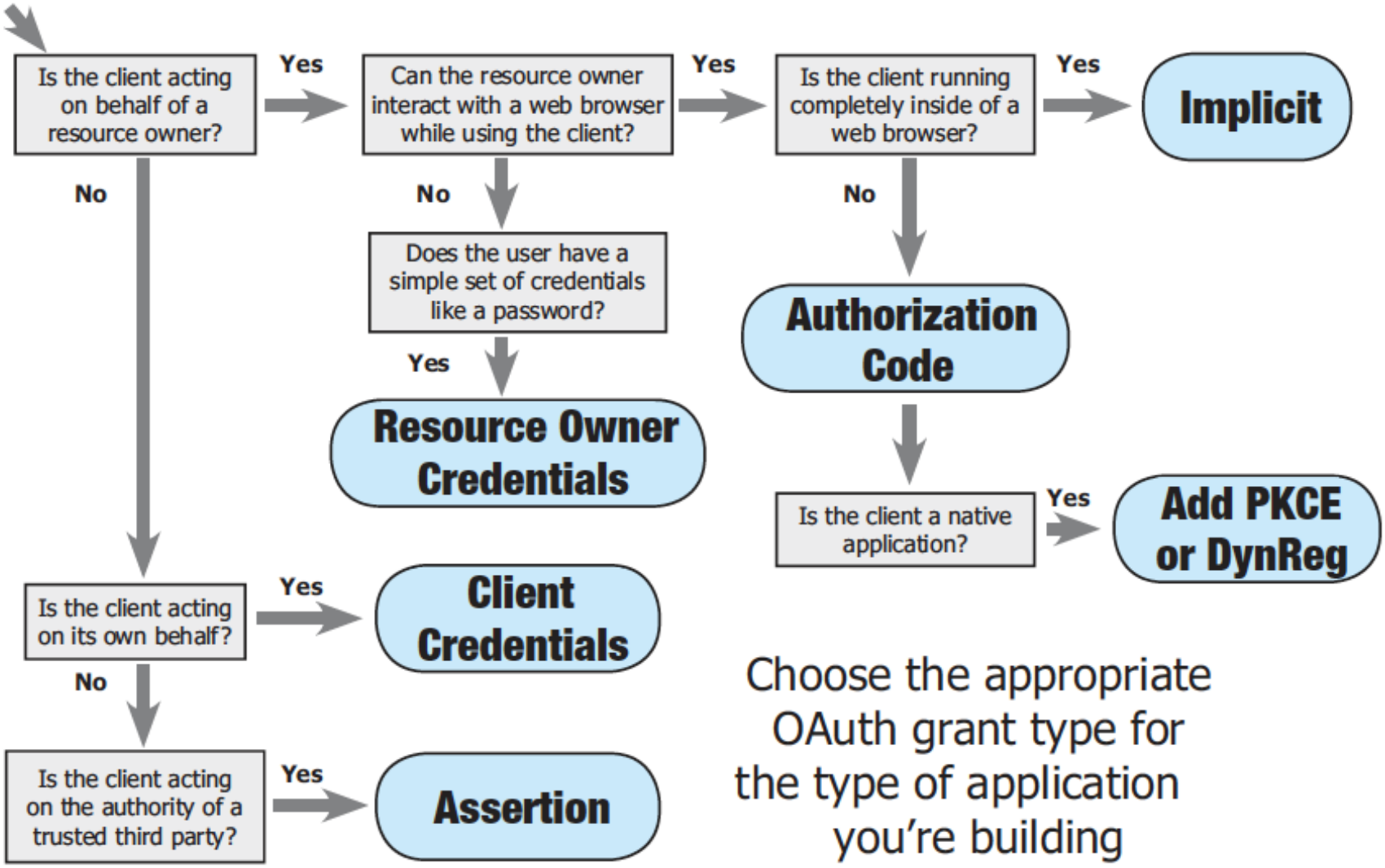
OAuth 2.0

- *assertation grant*
  - kada klijent ima već validan SAML/JWT „assertion“ od Identity Providera (IdP)
  - koraci
    1. klijent dobija digitalno potpisan assertion token (SAML ili JWT) iz eksternog IdP-a kojem se već autentifikovao (assertion=<SAML/JWT>)
    2. šalje ga autorizacionom serveru (grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer ili saml2-bearer)
    3. server verifikuje assertion i vraća access_token
  - upotreba
    - Single Sign-On (npr. ako je prethodno korisnik dobio SAML token prijavom kroz korporativni browser može isti da se iskoristi kroz mobilnu aplikaciju za dobijanje access_tokena)



OAuth 2.0

- koji tok (grant type) odabrati



JSON Web Token (JWT)

- JSON web token (JWT) je otvoreni standard (RFC 7519) koji definiše kompaktan i samostalan način za bezbedno prenošenje informacija između strana kao JSON objekta
- ove informacije mogu biti verifikovane i pouzdane jer su digitalno potpisane
- JWT-ovi mogu biti potpisani korišćenjem tajne (sa HMAC algoritmom) ili para javnih/privatnih ključeva koristeći RSA ili ECDSA

JSON Web Token (JWT)

- iako JWT-ovi mogu biti šifrovani da bi takođe obezbedili tajnost između strana, mi ćemo se fokusirati na potpisane tokene
- potpisani tokeni mogu da verifikuju integritet zahteva sadržanih u njima, dok šifrovani tokeni skrivaju te zahteve od drugih strana
- kada se tokeni potpisuju pomoću parova javnih/privatnih ključeva, potpis takođe potvrđuje da je samo strana koja drži privatni ključ ona koja ga je potpisala

JSON Web Token (JWT) – zašto koristiti?

- autorizacija
  - ovo je najčešći scenario za korišćenje JWT-a. Kada se korisnik prijavi, svaki naredni zahtev će uključivati JWT, omogućavajući korisniku da pristupi rutama, uslugama i resursima koji su dozvoljeni sa tim tokenom
  - Single Sign On (SSO) je funkcija koja danas široko koristi JWT, zbog malih troškova i mogućnosti da se lako koristi u različitim domenima
- razmena informacija
  - JSON web tokeni su dobar način za bezbedno prenošenje informacija između zainteresovanih strana
  - pošto JWT-ovi mogu biti potpisani, možemo biti sigurni da su pošiljaoci oni za koje kažu da jesu
  - pored toga, pošto se potpis izračunava pomoću zaglavlja i korisnog opterećenja, takođe možemo da proverimo da sadržaj nije menjan

JSON Web Token (JWT) – kako izgleda?

- u svom kompaktnom obliku, JSON web tokeni se sastoje od tri dela odvojena tačkama (.), a to su:
  - zaglavlje (engl. header)
  - sadržaj (engl. payload)
  - potpis (engl. signature)
- JWT struktura obično izgleda ovako:

```
xxxxx.yyyyy.zzzzz
```

JSON Web Token (JWT) – zaglavlje

- zaglavlje se obično sastoji od dva dela:
  - tip tokena, koji je JWT
  - algoritam heširanja ili šifrovanja koji se koristi, kao što je HMAC SHA256 ili RSA
- na primer:

```
{
    "alg": "HS256",
    "typ": "JWT"
}
```

- zatim, ovaj JSON je Base64Url kodiran da formira prvi deo JWT-a

JSON Web Token (JWT) – sadržaj

- drugi deo tokena je korisni sadržaj, koji sadrži tvrdnje (engl. claims)
- tvrdnje su izjave o entitetu (obično korisniku) i dodatni podaci
- postoje tri vrste zahteva:
  - registrovane tvrdnje
  - javne tvrdnje
  - privatne tvrdnje

```
{
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
}
```

- zatim, ovaj JSON je Base64Url kodiran da formira drugi deo JWT-a

JSON Web Token (JWT) – sadržaj

- registrovane tvrdnje:
  - ovo je skup unapred definisanih tvrdnji/zahteva koji nisu obavezni, ali preporučeni, da bi se obezbedio skup korisnih, interoperabilnih tvrdnji
  - neki od njih su: iss (izdavač), exp (vreme isteka), sub (predmet), aud (publika) i drugi

- javne tvrdnje:
  - mogu biti definisani po želji onih koji koriste JWT
  - ali da bi se izbegle kolizije, trebalo bi da budu definisani u IANA JSON Web Token Registry ili da budu definisani kao URI koji sadrži imenski prostor otporan na kolizije

- privatne tvdnje:
  - ovo su prilagođene tvrdnje stvorene za deljenje informacija između strana koje se slažu da ih koriste i nisu ni registrovane ni javne tvdnje

JSON Web Token (JWT) – potpis

- da bi se kreirao deo potpisa, mora se uzeti kodirano zaglavlje, kodirani sadržaj, tajna, algoritam naveden u zaglavlju i potpisati sve to
- na primer, ako se koristi HMAC SHA256 algoritam, potpis će biti kreiran na sledeći način:

```
HMACSHA256(
        base64UrlEncode(header) + "." +
        base64UrlEncode(payload),
        secret)
```

- potpis se koristi da se potvrdi da poruka nije promenjena usput, a u slučaju tokena potpisanih privatnim ključem, takođe može da potvrdi da je pošiljalac JWT-a onaj za koga kaže da jeste

JSON Web Token (JWT) – ceo token

- izlaz su tri Base64-URL stringa razdvojena tačkama koji se lako mogu prosleđivati u HTML i HTTP okruženjima, dok su kompaktniji u poređenju sa standardima zasnovanim na XML-u kao što je Security Assertion Markup Language (SAML)
- sledeći primer prikazuje JWT koji ima prethodno kodirano zaglavlje i sadržaj i potpisan je tajnom:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

JSON Web Token (JWT) – autentifikacija

- u autentifikaciji, kada se korisnik uspešno prijavi koristeći svoje kredencijale, JSON web token će biti vraćen i mora biti sačuvan lokalno (obično neadekvatno u lokalnom skladištu???!!!???, mogu se koristiti i kolačići), umesto tradicionalnog pristupa kreiranja sesije na serveru i vraćanja kolačiča
- kad god korisnik želi da pristupi zaštićenoj ruti, trebalo bi da pošalje JWT, obično u zaglavlju autorizacije koristeći šemu nosioca
- stoga bi sadržaj zaglavlja trebao izgledati ovako:

```
Authorization: Bearer <token>
```

JSON Web Token (JWT) – autentifikacija

- ovo je mehanizam autentifikacije bez stanja (engl. stateless) pošto se korisničko stanje nikada ne čuva u memoriji servera
- zaštićene rute servera će proveriti da li postoji važeći JWT u zaglavlju autorizacije, a ako postoji, korisniku će biti dozvoljen pristup
- pošto su JWT-ovi samostalni, sve potrebne informacije su tu, smanjuje se potreba za vraćanjem nazad i napred do baze podataka
- ovo omogućava da se u potpunosti osloni na API-je podataka koji nemaju stanje, pa čak i da se upućuju zahtevi ostalim servisima
- nije važno koji domeni opslužuju API-je, jer *Cross-Origin Resource Sharing* (CORS) neće biti problem jer se ne koriste kolačići

JSON Web Token (JWT) – Token Sidejacking

- dešava se kada je napadač ukrao token i koristi ga za pristup sistemu koristeći ciljani korisnički identitet
- prevencija:
  - dodavanje „korisničkog konteksta“ u token:
    - nasumični string se generiše tokom faze autentifikacije
      - šalje se klijentu kroz ojačani kolačić - HttpOnly + Secure + SameSite + Max-Age + prefiksi kolačića
      - izbegavati podešavanje zaglavlja za istek tokena tako da se kolačić briše kada se pretraživač zatvori
      - postaviti Max-Age na vrednost manju ili jednaku vrednosti isteka JWT tokena, ali nikada veću
    - SHA256 heš nasumičnog stringa biće uskladišten u tokenu (umesto neobrađene (raw) vrednosti) kako bi se sprečili problemi sa XSS-om koji bi omogućili napadaču da pročita nasumične vrednosti stringa i podesi očekivani kolačić
    - tokom provere tokena, ako primljeni token ne sadrži pravi kontekst kao u kolačiću, odbija se

JSON Web Token (JWT) – None algoritam za heširanje

- dešava se kada napadač promeni token i promeni algoritam heširanja da bi ukazao, preko ključne reči none, da je integritet tokena već verifikovan
  - neke biblioteke tretiraju tokene potpisane algoritmom none kao važeće, tako da napadač može da promeni zahteve za tokene i aplikacija će i dalje imati poverenje u modifikovani token
- prevencija:
  - koristiti JWT biblioteku koja nije izložena ovoj ranjivosti
  - tokom provere tokena, izričito proveriti da se koristi očekivani algoritam

```
{
    "alg": ”none",
    "typ": "JWT"
}
```

JSON Web Token (JWT) – tajna u tokenu

- kada je token zaštićen HMAC algoritmom, bezbednost tokena zavisi isključivo od jačine tajne koja se koristi
  - ako napadač dođe do JWT tokena, može da izvrši offline napad i pokuša da otkrije tajnu grubom silom
  - ako je napad uspešan, može da modifikuje token i ponovo ga potpiše pomoću ključa koji je dobijen
- mitigacija:
  - tajna treba da ima najmanje 64 znaka i da se generiše nasumično
  - alternativno, koristiti tokene koji su potpisani sa RSA

JSON Web Token (JWT) – važenje tokena

- token postaje nevažeći tek kada istekne
  - korisnik nema ugrađenu funkciju da eksplicitno opozove valjanost tokena u slučaju krađe
- mitigacija:
  - dobro implementirano rešenje za Token Sidejacking izbegava potrebu za održavanjem liste povučenih tokena na strani servera
    - ako i kolačić nije ukraden, JWT je neupotrebljiv
  - drugi način zaštite od ovoga je implementacija liste povučenih tokena
    - sadrži heš tokena sa datumom opoziva
    - unos mora ostati u listi najmanje do isteka tokena

JSON Web Token (JWT) – čuvanje tokena na klijentu

- problem ako aplikacija skladišti token u:
  - localStorage pretraživača
  - kolačiću i dostupan je JavaScript kodu (potencijalni XSS napad)
- mitigacija:
  - čuvati token bar u sessionStorage pretraživača ili koristiti JavaScript closures sa privatnim promenljivama
  - dodati ga kao Bearer HTTP Authentication zaglavlje
  - dodati fingerprinting informacije u token (kao kod Token Sidejacking mitigacija)
    - ako se čuva u sessionStorage i dalje je podložan XSS napadu ali dodavanjem fingerprinting informacija sprečava se ponovna upotreba tokena
    - dodati Content Security Policy[1] zaglavlja za dodatnu bezbednost
    - alternativa je korišenje JavaScript closures gde se svi zahtevi rutiraju kroz JavaScript modul koji enkapsulira token u privatnu promenljivu kojoj se može pristupiti samo iz tog modula

[1] Content Security Policy OWASP - https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

Poverenje u sisteme za autentifikaciju

- Ken Thompson: Reflections on Trusting Trust
  - govor povodom prijema Tjuringove nagrade
  - opisuje korišćenje samo-izmenjujućih programa za postavljanje zadnjih vrata (*backdoor*)

- korak 1: izmeni C kompajler tako da se, prilikom prevođenja UNIX login programa, za određenu lozinku automatski dodele korisniku maksimalne privilegije
  - ova izmena se lako uočava pregledom izvornog koda kompajlera
- korak 2: izmeni C kompajler tako da se, prilikom prevođenja samog sebe, automatski ugradi „trojanac“ iz koraka 1
  - teže za detekciju: niko ne proverava izvorni kod već prevedenog kompajlera!
- korak 3: ukloni sve „trojanske“ izmene iz izvornog koda kompajlera i prevedi ga ponovo
  - upravo prevedeni kompajler i dalje sadrži backdoor, iako ga više nema u vidljivom kodu

- zaključak: ne možemo verovati nijednom programu koji nismo sami napisali
  - naročito ne programima koje proizvode kompanije koje zapošljavaju ljude kao što je K.T. ;)
  - "trojanca" je moguće podmetnuti i u disasembler tako da se izmena ne može videti ni inspekcijom prevedenog koda
  - jedino potpuno sigurno rešenje: sami napisati i kompajler i disasembler… ali ko to radi danas?;)