Implementacija sigurnosnih protokola za razmjenu tajnog ključa

Živko, Andrija

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Andrija Živko

Implementacija sigurnosnih protokola za razmjenu tajnog ključa

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Andrija Živko

Matični broj: 16143755–R

Studij: Informacijski sustavi

Implementacija sigurnosnih protokola za razmjenu tajnog ključa

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Ivan Magdalenić

Varaždin, rujan 2023.

Andrija Živko

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema završnog rada bazirana je na implementaciji sigurnosnih protokola pomoću kojih se odvija razmjena tajnog ključa kod komunikacije između uređaja. Teoretski dio rada više je posvećen tome kako su implementirani sigurnosni protokoli za razmjenu tajnog ključa u mrežnim protokolima koji se danas najčešće koriste. Prije same obrade glavne teme, čitatelja se upoznaje s osnovnim konceptima koje bi trebao savladati kako bi mogao razumjeti na čemu se glavna tema rada bazira. Pojmovi koji se ukratko spominju i obrađuju su kriptografija, kriptosustav, simetrični i asimetrični kriptosustavi te neki sigurnosni protokoli. Cilj rada je upoznati čitatelja s pojmovima koji se koriste kod komuniciranja između uređaja preko interneta te dati do znanja koji koraci se najčešće odvijaju kod slanja svake poruke. Praktični dio rada baziran je na primjeru uspostave komunikacije između dva korisnika. Prikazuje se način na koji se uspostavljaju sigurnosni protokoli prije nego li korisnici krenu razmjenjivati poruke koje ne bi željeli da ostanu javne.

Ključne riječi: kriptografija; kriptosustav; Diffie-Hellman; Needham-Schroeder; TLS; IPsec; SSH; Linux; Wireshark; Visual Studio Code; C++;

Sadržaj

1. Uvod

Tema ovog završnog rada je „Implementacija sigurnosnih protokola za razmjenu tajnog ključa“. Svakidašnjim pretraživanjem interneta primamo i šaljemo podatke preko raznih mreža, čak i kada tog nismo svjesni. Podaci se šalju čak i kada nema našeg aktivnog prisustva te se među tim podacima nalaze i neke osjetljive informacije u koje ne bismo željeli da drugi imaju uvid. Neovisno o tome gdje se na internetu nalazimo ili kakve stranice posjećujemo, uvijek ćemo negdje upisati svoje lozinke ili neke druge podatke i to je veliki razlog zašto uopće postoji sektor za sigurnost na internetu. Koncept tajnog ključa kod slanja poruka je tako napravljen da riješi navedeni problem kod slanja podataka. Tajnim ključem želimo da podatke može vidjeti osoba koja za to ima autoritet.

Temu sam odabrao jer želim povećati svoje znanje u sektoru kibernetičke sigurnosti te savladati nove koncepte. Cilj ovog završnog rada je upoznavanje čitatelja s konceptom tajnih ključeva kod razmjene podataka na internetu te grupiranju istih prema određenim kriterijima. Osim toga, prikazat će se kako su implementirani takvi sigurnosni protokoli u mrežnim protokolima raznih slojeva OSI modela. Praktičnom komponentom čitatelja se upoznaje s konceptima u praktičnom pogledu i na što jednostavniji način mu se prikazuju implementacije određenih protokola. Tehnologije koje su korištene prilikom izrade rada su virtualno okruženje Vmware, Kali distribucija Linux operacijskog sustava, Wireshark, Microsoft Visual Studio Code te programski jezik C++ i Linux Bash za kompajliranje i pokretanje programskih rješenja.

2. Metode i tehnike rada

Prilikom izrade ovog rada pročitao sam i proučio poglavlja u udžbeniku navedenom u literaturi koja su povezana s temom. Osim knjige koristio sam i proučavao razne web stranice na internetu koje su mi pružile bitne informacije koje sam iskoristio za obradu nekih teoretskih poglavlja ovog rada. Kako bih još više nadopunio svoje znanje o samoj temi, proučavao sam razne video materijale na internetu koje su izradili ljudi s poprilično velikim znanjem iz područja sigurnosti. U praktičnom dijelu, operacijski sustav Linux sam pokretao u virtualnom okruženju putem aplikacije VMware. Kao programske alate koristio sam Wireshark, Visual Studio Code te programski jezik C++. Svako programsko rješenje kompajlirao sam putem GCC-a (engl. GNU Compiler Collection).

3. Kriptografija

Napadač može raznim tehnikama uhvatiti podatke koje šaljemo internetom ili se lažno predstaviti i slati poruke kao neovlaštena osoba. Kriptiranjem pokazat će se da se mogu zadovoljiti svi sigurnosni zahtjevi, osim raspoloživosti koja ovisi o drugim aspektima, kod komuniciranja između umreženih računala. Prisluškivanje nije moguće spriječiti u svim slučajevima te se zbog toga koristi kriptografija. Kriptiranjem podataka čitljiv podatak koji napadaču može dati razumljivu informaciju činimo nerazumljivim. U obrnutom slučaju, kada ovlašteni korisnik želi pročitati podatak, vrši se dekriptiranje podataka kako bi se pretvorio u jasni tekst [1].

Današnji kriptografski protokoli mnogo su složeniji od nekih starijih koji su se zasnivali na zamjeni znakova i nekih drugih pravila. Koristimo matematičke funkcije, odnosno algoritme kojima se nizovi bitova jasnog teksta pretvaraju u nizove bitova kriptiranog teksta [1].

Funkcija kriptiranja može se zapisati u sljedećem obliku:

$$C = E(P, K_E)$$

P – jasni tekst

C – kriptirani tekst

E – funkcija kriptiranja

K_E – parametar ili ključ kriptiranja

Funkciju dekriptiranja ćemo zapisati:

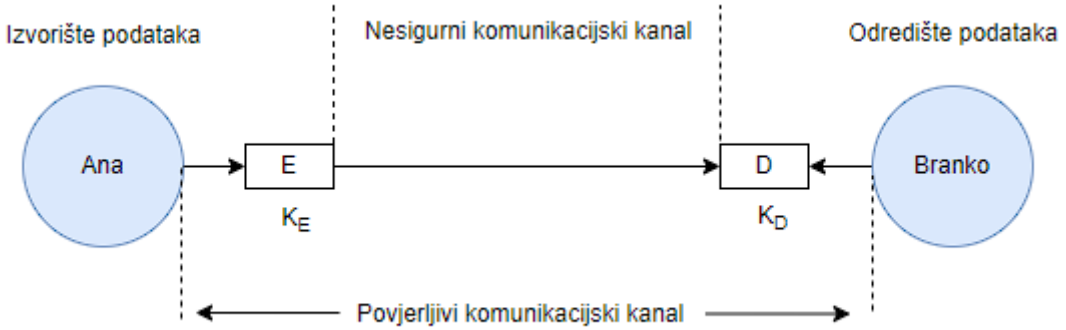
$$P = D(C, K_D)$$

D – funkcija dekriptiranja

K_D – parametar ili ključ dekriptiranja

3.1. Kriptosustav

Funkcija kriptiranja E i dekriptiranja D čine kriptosustav. Na primjerima korisnika Ana i Branko pokazat će se kako detaljnije kriptosustav funkcionira kod upotrebe.



Slika 1: Pojednostavljeni prikaz kriptosustava (Prema: Budin, Golub, Jakobović i Jelenković, 2016)

Na slici se mogu vidjeti dva korisnika, Ana i Branko, koji komuniciraju preko komunikacijskog kanala. Ana obavlja kriptiranje teksta svojim ključem K_E. Kriptirana se poruka tada šalje preko komunikacijskog kanala Branku. Poruka je putem kriptirana te je time zaštićena od napadača. Ukoliko poruka ne bi bila kriptirana, ne bi bila zaštićena od uljeza jer putuje nesigurnim komunikacijskim kanalom. U trenutku kada Branko primi poruku od Ane, dekriptira je svojem K_D ključem te kao rezultat dobiva jasan tekst. Ako pretpostavimo da jedino Branko zna ključ dekriptiranja, cijeli ovaj komunikacijski kanal nazivamo povjerljivi komunikacijski kanal. Ukoliko i napadač prisluškuje i dobije poruku, neće je moći dekriptirati jer nema ključ dekriptiranja te za njega ta poruka ne predstavlja ništa. Dobrotu kriptosustava određujemo težinom otkrivanja ključa dekriptiranja. Dakle time smo objasnili da kriptosustav služi za pretvaranje nesigurnog komunikacijskog kanala u povjerljivi [1].

U praksi koristimo dvije vrste kriptosustava:

- Simetrični kriptosustavi
- Asimetrični kriptosustavi

Simetrični kriptosustavi su karakterizirani po tome da imaju jedan ključ kriptiranja K, odnosno ključ kriptiranja i dekriptiranja su jednaki. Opisat ćemo ga sljedećim izrazima:

$$C = E(P, K)$$

$$P = D(C, K)$$

$$P = D(E(P, K), K)$$

Asimetrični kriptosustavi su karakterizirani po tome da imaju različite ključeve kod kriptiranja i dekriptiranja te ih možemo opisati sa već navedenim izrazima u prošlom poglavlju.

3.1.1. Simetrični kriptosustavi

Kao što je već napomenuto, simetrični kriptosustavi koriste isti ključ K i za kriptiranje i za dekriptiranje. Zahtjev da obje strane kod komunikacije imaju pristup tajnom ključu jedan je od glavnih nedostataka šifriranja simetričnim ključem, u usporedbi s asimetričnim kriptosustavima. Međutim, algoritmi koji koriste simetrične kriptosustave obično su bolji za veće enkripcije. Imaju manju veličinu ključa, što znači da zauzimaju manje prostora za pohranu i omogućavaju brži prijenos. To je i razlog zašto se enkripcija asimetričnim ključem često koristi za razmjenu tajnog ključa za šifriranje simetričnim ključem [2].

Simetrični kriptosustavi mogu koristiti šifre toka ili šifre blokova. Šifre toka šifriraju znamenke ili slova poruke jednu po jednu (npr. ChaCha20). Supstitucijske šifre su dobro poznate šifre, ali se mogu lako dešifrirati korištenjem frekvencijske tablice. Šifre blokova uzimaju neki broj bitova te ih šifriraju u jednu jedinicu, popunjavajući jasan tekst kako bi se postigla višestruka veličina bloka. Jedan od takvih primjera bi bio AES i on koristi 128-bitne blokove [2].

3.1.2. Asimetrični kriptosustavi

Asimetrični kriptosustavi ili kriptografija s javnim ključem koristi parove povezanih ključeva. Svaki par ključeva sastoji se od javnog ključa i privatnog ključa. Parovi ključeva generiraju se kriptografskim algoritmima koji se temelje na matematičkim problemima (npr. RSA). Sigurnost

kriptografije s javnim ključem ovisi o očuvanju tajnosti privatnog ključa. Javni ključ slobodno se može objaviti javnosti i time ne dolazi do narušavanja sigurnosti [3].

Kod asimetričnim kriptosustava, bilo tko s javnim ključem može kriptirati poruku, ali samo oni koji imaju privatni ključ mogu dekriptirati kriptirani tekst kako bi dobili jasan tekst. Međutim, enkripcija s javnim ključem ne skriva metapodatke poput toga koje je računalo izvor upotrijebilo za slanje poruke, kada ju je poslao ili koliko je duga. Enkripcija javnim ključem ne otkriva primatelju ništa o tome tko je poslao poruku već ona samo skriva sadržaj poruke u kriptiranom tekstu koji se može dekriptirati privatnim ključem primatelja. Ako uz asimetrične kriptosustave koristimo i sustave s digitalnim potpisom, tada svatko s odgovarajućim javnim ključem može provjeriti odgovara li potpis poruci. Asimetrični kriptosustavi se danas koriste u internet protokolima kao što su TLS i SSH [3].

4. Sigurnosni protokoli

Sigurnosni protokoli za razmjenu tajnog ključa predstavljaju skup pravila i koraka koji se odvijaju tijekom ostvarivanja veze između dva klijenta ili klijenta i servera. Rezultat takvih protokola jest dogovoreni tajni ključ s obje strane komunikacijskog kanala pomoću kojeg mogu kriptirati poruke raznim algoritmima za kriptiranje te dekriptirati nadolazeće poruke. Razmjena ključa se može vršiti na mnogo načina no sada će se prikazati teoretski koncept dva različita protokola za razmjenu tajnog ključa. U praktičnom dijelu rada će se implementirati četiri protokola od kojih su dva ista koje će se u sljedećim poglavljima obraditi na razini teorije.

4.1. Diffie-Hellmanov postupak za razmjenu tajnog ključa

Diffie-Hellmanova razmjena ključeva matematička je metoda digitalne enkripcije koja sigurno razmjenjuje kriptografske ključeve između dviju strana preko javnog kanala bez da se njihov razgovor prenosi preko interneta. Komponente ključeva se nikad ne prenose izravno, što napadaču predstavlja problem. Dvije strane koje će stati u komunikaciju do tad nisu imali prethodnog znanja jedna o drugoj, ali zajedno stvaraju ključ. Cilj Diffie-Hellmana je sigurno uspostavljanje kanala za stvaranje i dijeljenje ključa za algoritme koji koriste simetričan ključ [4].

Sudionici u komunikaciji unaprijed se slože oko dva vrlo velika broja n i g, ali takva da je g relativno prost u odnosu na n, odnosno to znaći da im najveći zajednički djelitelj mora biti jednak 1. Navedeni brojevi ne moraju biti tajni. Za primjer ćemo koristiti ranije navedene korisnike Ana i Branko [1].

1. Ukoliko Ana želi uspostaviti komunikaciju s Brankom, odabire veliki prirodni broj x i šalje branku rezultat kojeg izračuna sljedećom jednadžbom:

$$X = g^x mod\ p$$

2. Branko tada ponovi isti postupak s nasumičnim velikim prirodnim brojem y i šalje Ani rezultat.
3. Ana u sljedećem koraku računa:

$$K = Y^x mod\ p = (g^y)^x mod\ p = g^{xy} mod\ p$$

4. Branko ponavlja sličan postupak:

$$K = X^y mod\ p = (g^x)^y mod\ p = g^{xy} mod\ p$$

5. Postupak završava te smo kao rezultat dobili ključ K koji Ana i Branko mogu koristiti za simetrično kriptiranje.

4.2. Needham-Schroederov protokol

Needham-Schroederov protokol baziran na simetričnom kriptosustavu je sigurnosni protokol razvijen od Rogera Needhama i Michaela Schroedera. Cilj ovog protokola je ostvariti sesijski ključ za zaštitu daljnje komunikacije. Kako bi se ovaj protokol objasnio na primjeru, za primjer će se uzeti već poznati klijenti Ana i Branko. KDC označava centar za distribuciju ključeva kojem oboje korisnika vjeruju. Taj centar ima ključeve od oba korisnika Ane i Branka kako bi mogao dekriptirati njihove poruke [1].

Prvi korak u ovom protokolu označuje poruku P_1 koju šalje Ana u kojoj se nalazi neki kod zahtjeva za dodjelu ključa, identifikator Ane te identifikator Branka. Tako KDC zna koji ključ K_{AB} treba stvoriti. Poruka nije u poslana u jasnom obliku već je Ana kriptira svojim ključem K_A [1].

U drugom koraku KDC prima poruku od Ane te je dekriptira ključem K_A koji ima u svojoj listi ključeva. Brankovim ključem K_B kriptira identifikator Ane ID_A i ključ K_{AB}. Cijela tu poruku ćemo nazvati M_1. Nakon toga stvara poruku P_2 tako da cijelu M_1 poruku zajedno sa kodom zahtjeva Z_A, identifikatorom Ane ID_A i ključom K_{AB} kriptira s Aninim ključem K_A [1].

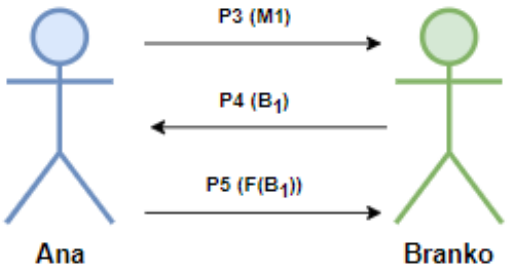
Slika 2: Komunikacija između Ane i KDC-a

Ana u trećem koraku prima poruku P_2 od KDC-a te je dekriptira svojim ključem K_A. Pronalazi ključ K_{AB} kojeg će koristiti za komunikaciju s Brankom te šalje branku poruku P_3 koja je zapravo poruka M_1 koju je dobila porukom P_2 [1].

Branko tada dekriptira poruku P_3 i tako i on ima ključ K_{AB}. Sada oba korisnika imaju simetričan ključ K_{AB} no Branko još želi provjeriti da Ana uistinu ima isti ključ K_{AB} te stvara poruku P_4 tako da generira slučajan broj B_1 (engl. nonce) koji kriptira ključem K_{AB} i šalje Ani [1].

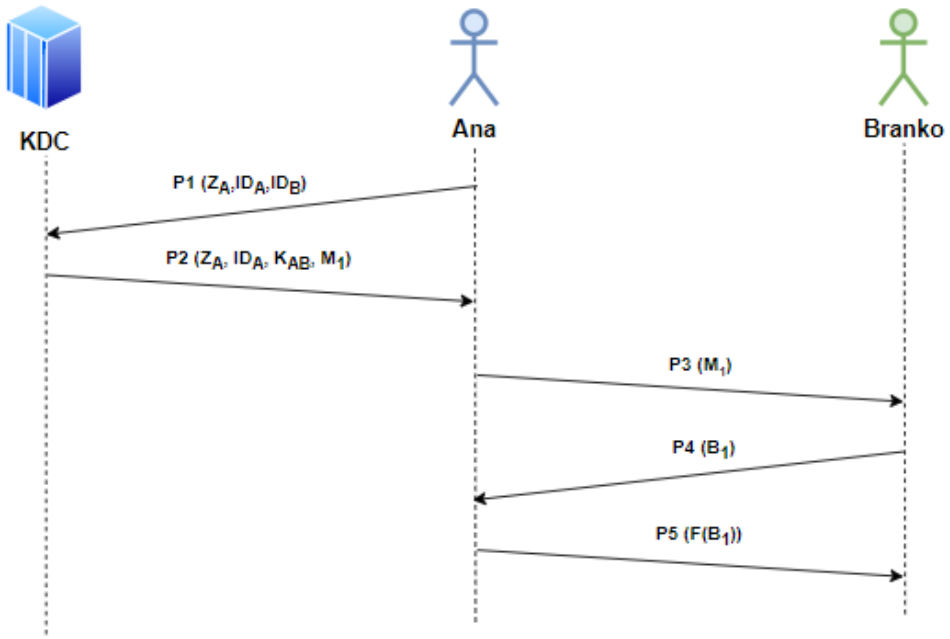
Ana dekriptira poruku P_4 te obavlja jednostavnu operaciju na broju B_1 kojeg je zaprimila i šalje ga natrag u poruci P_5 koju je kriptirala s K_{AB} ključem kako bi pokazala da je još uvijek „živa“ i da uistinu ima ključ K_{AB} [1].

Branko zaprima poruku i provjerava vraća li inverz operacije kojom je Ana izračunala novi broj isti broj koji je on generirao.



Slika 3: Komunikacija između Ane i Branka

Sljedeća slika grafički prikazuje cijeli postupak, odnosno tijek poruka između Ane, Branka i KDC-a.



Slika 4: Tijek poruka u Needham-Schroeder protokolu (Prema: Budin, Golub, Jakobović i Jelenković, 2016)

5. Sigurnosni protokoli za razmjenu tajnih ključeva u mrežnim protokolima

Do sad su u radu obrađena dva protokola koja se mogu koristiti kod razmjene tajnih ključeva. Diffie-Hellman protokol će se još par puta spomenuti jer se koristi kod mnogih mrežnih protokola u svojoj originalnoj formi ili u formi kojoj se ključevi računaju uz pomoć odabranih točaka na eliptičnoj krivulji. Na primjerima nekih mrežnim protokola koji operiraju na više slojeva OSI modela objasnit će se proces razmjene ključeva kako bi uspostavili sigurnu komunikaciju.

5.1. TLS

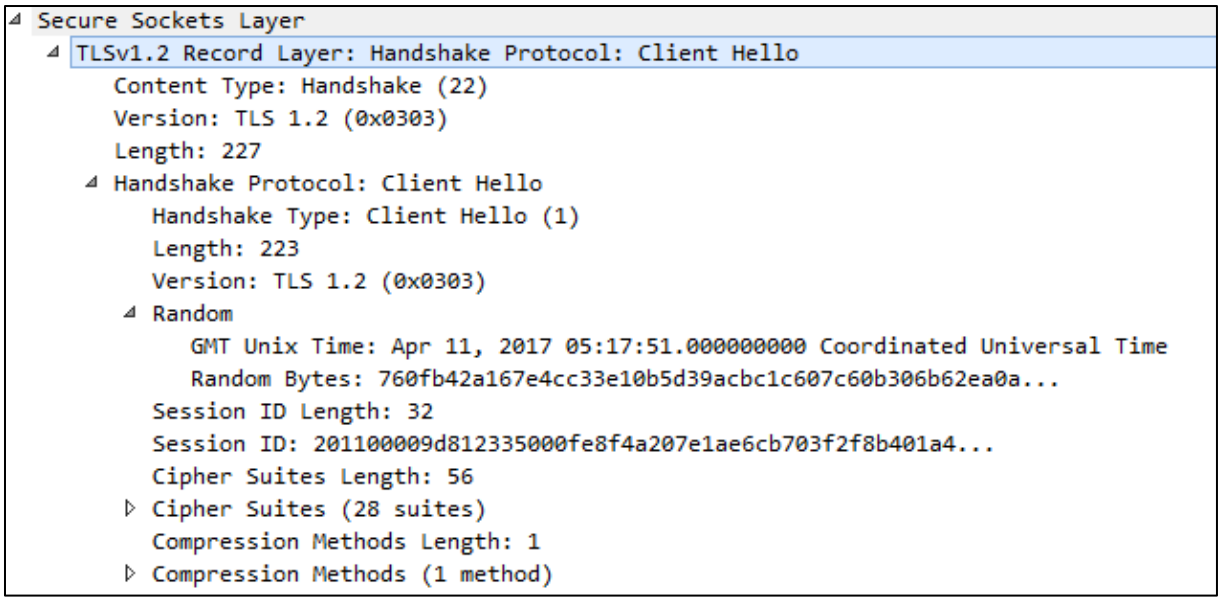
Kada pretražujemo internet često se može vidjeti HTTPS oznaka kod adrese stranice koja se učitava. To je sigurnija odnosno kriptirana verzija HTTP protokola koja koristi TLS mrežni protokol. U sljedećem paragrafu će se opisati kako zapravo izgleda TLS rukovanje (handshake).

5.1.1. TLS handshake

Client hello označuje prvi korak kojim klijent uspostavlja prvi kontakt sa serverom [5].

Klijent šalje:

- Informaciju o najvećoj verziji TLS-a koju podržava
- Nasumično generirani broj (32 bajta)
- ID sesije (8 bajta)
- Listu Ciper suit-a koje podržava



Slika 5: Prvi korak kod uspostave TLS-a (Client Hello)

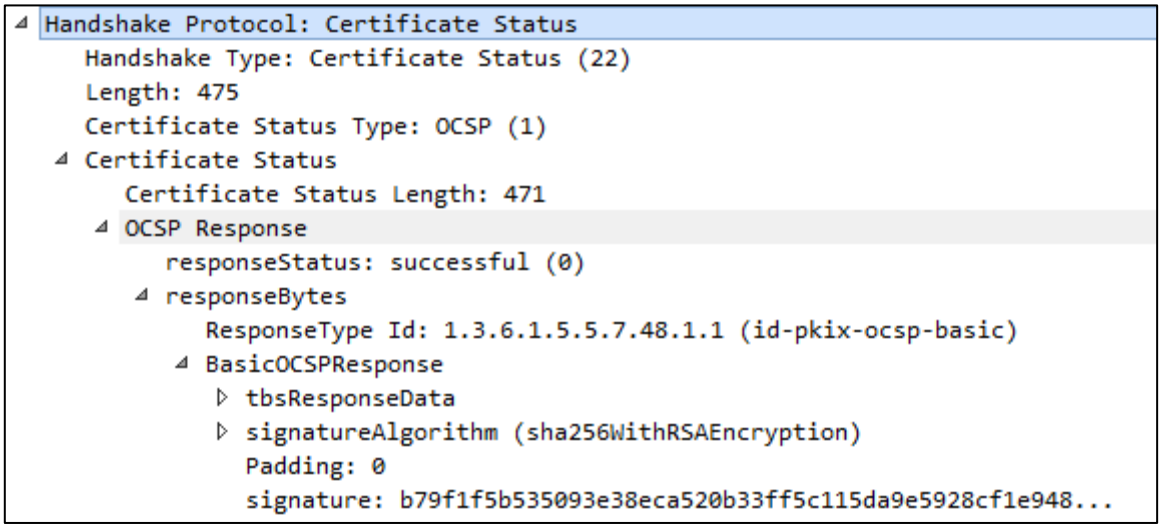
Drugi korak koji se odvija je Server hello odnosno prvi kontakt servera s klijentom u kojem će server poslati iste tipove informacija koje je poslao i klijent u prošloj poruci [5]. Za primjer će se uzeti da je server odabrao TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 cipher suite. Ta informacija klijentu znači da će se za kriptiranje podataka koristiti Diffie-Hellman Ephemeral protokol za razmjenu ključa te AES sa SHA-256 hash algoritmom za kriptiranje podataka.

Slika 6: Drugi korak kod uspostave TLS-a (Server Hello)

Nakon što je server poslao Server hello, šalje certifikat i cijeli lanac certifikata. Time klijent dobiva javni ključ koji će dalje koristiti za kriptiranje podataka koje će slati serveru [5].

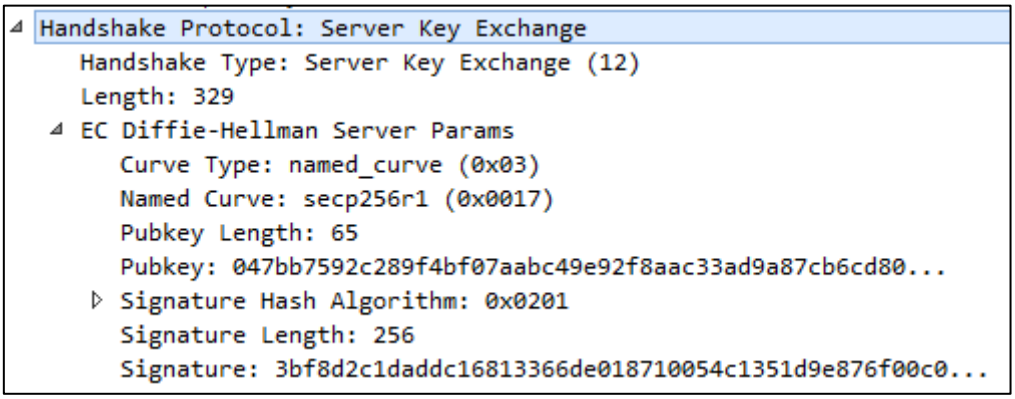
Slika 7: Lanac certifikata koje je server poslao klijentu

Klijent ujedno mora provjeriti da certifikat nije istekao. Zatražuje kopiju certifikata od CA-a. Također, provjerava digitalni potpis dekriptiranjem SHA-256 hash-a s javnim ključem [5].

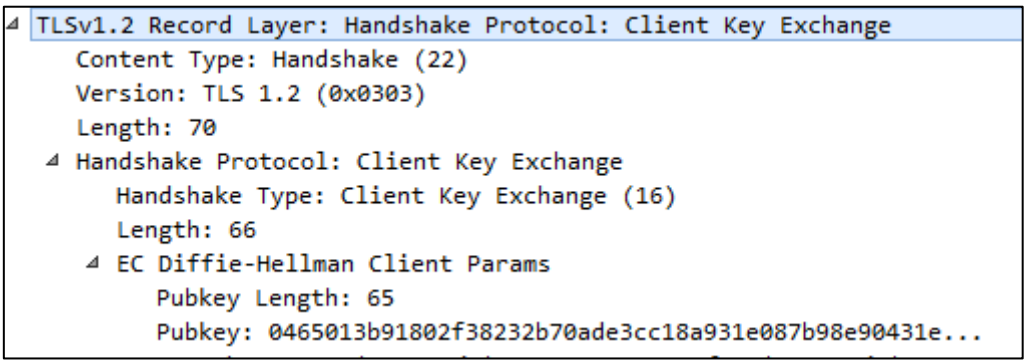


Slika 8: Provjera valjanosti certifikata od strane klijenta

Klijent i poslužitelj moraju se dogovoriti o zajedničkoj pre-master tajni (K) prije nego što mogu slati kriptirane poruke međusobno. To se može postići korištenjem Diffie-Hellman protokola. K se zatim može unijeti kroz HMAC funkciju proširenja ključa za stvaranje ključeva sesije [5].



Slika 9: Razmjena ključa (Server)



Slika 10: Razmjena ključa (Klijent)

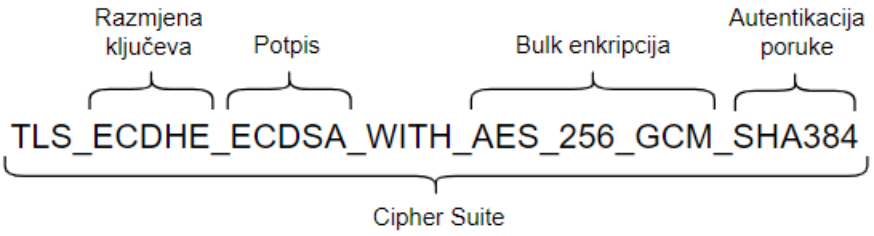
Kada se stvori master tajna, klijent šalje ClientFinish, a zatim poslužitelj odgovara sa ServerFinish. U ovoj točki asimetrična kriptografija prestaje i preuzima je simetrična kriptografija jer i klijent i poslužitelj znaju K [5].

5.1.2. Cipher suite u TLS-u

Cipher suite je skup kriptografskih algoritama. Kod implementacije TLS protokola koristimo algoritme iz ciper suit-a za stvaranje ključeva, šifriranje informacija te razmjene ključeva.

Algoritmi za razmjenu ključeva štite informacije potrebne za stvaranje zajedničkih ključeva. Osim algoritama za razmjenu ključeva koristimo algoritme za enkripciju podataka te autentikaciju poruka. [6]

Primjer cipher suite-a možemo označiti sljedećim stringom:



Slika 11: Primjer Cipher suite-a (Prema: Microsoft, 2023.)

U TLS-u algoritmi za razmjenu ključeva specificiraju razmjenu simetričnih ključeva potrebnih za proces šifriranja. Budući da se veza smatra nesigurnom, mora se implementirati zaseban asimetrični proces dolaska do ključeva kako bi se zajamčilo da će samo dvije strane koje se povezuju imati simetrične ključeve. [6]

Za svaki cipher suite može se provjeriti njegova sigurnost u bazama znanja na internetu. Cipher suite koji je naveden na slici iznad ocijenjen je sa A+ ocjenom iz razloga što je svaki pojedini algoritam koji se navodi ocijenjen A+ ocjenom.

5.2. IPsec

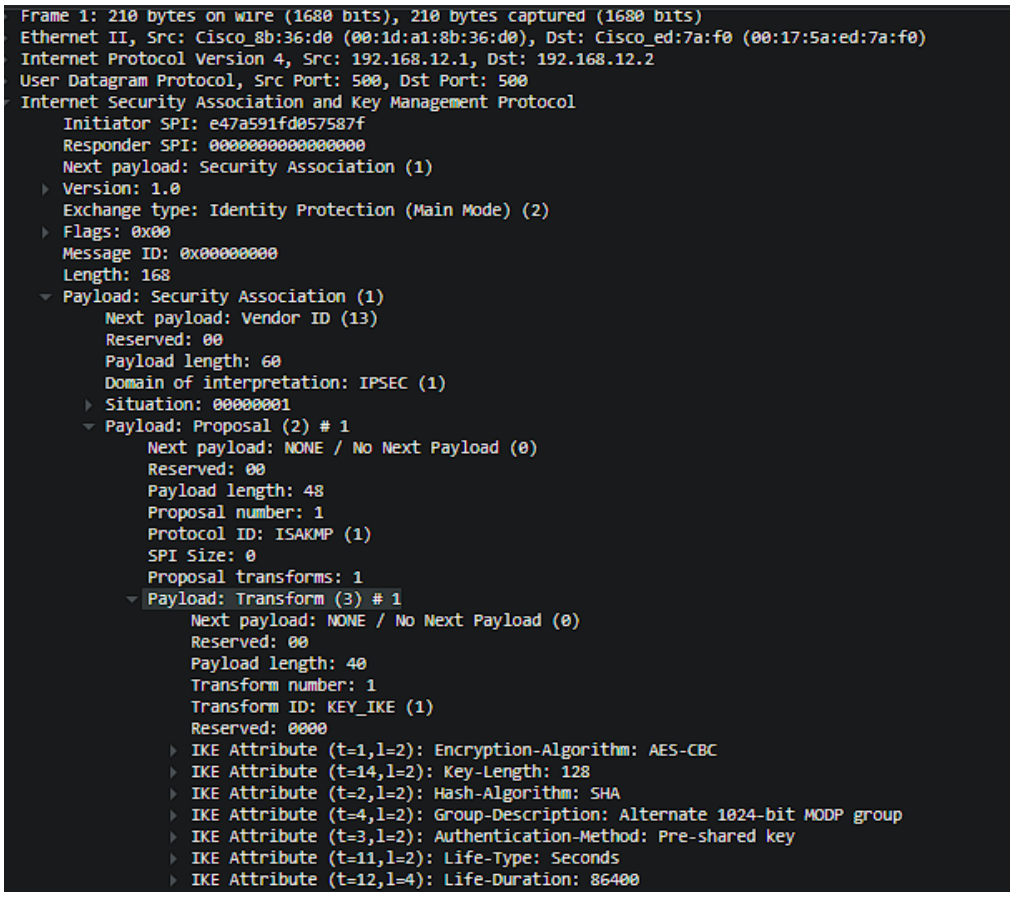
IPsec (Internet Protocol Security) je okvir koji pomaže zaštititi IP promet na mrežnom sloju. Njegovi čestu upotrebu možemo uočiti kod postavljanja VPN mreža. Zbog toga što je okvir, koristi skup protokola kako bi se omogućila povjerljivost, integritet, autentikacija te neponovljivost (anti-replay). Protokoli koji su korišteni kod uspostave IPsec suite-a su Authentication Header (AH), Encapsulating Security Protocol (ESP) i Security Association (SA). Security Association protokol se zapravo odnosi na razne protokole i algoritme koji se koriste za razmjenu tajnih ključeva. [8],[9]

Kako bi se zaštitili IP paketi, dva klijenta koja koriste IPsec moraju stvoriti IPsec tunel. Taj tunel je implementiran korištenjem protokola koji se naziva IKE (Internet Key Exchange) i sluša na portu 500 i 4500. IKE protokol jedan je od najčešćih SA protokola. Postoje dvije verzije ovog protokola, IKEv1 te IKEv2. IKEv2 podržava EAP autentikaciju (uz unaprijed dijeljenje ključeve i digitalne certifikate). Verzije nisu međusobno kompatibilne zbog većih promjena koje su se dogodile kod nadogradnje. [9]

5.2.1. IKEv1

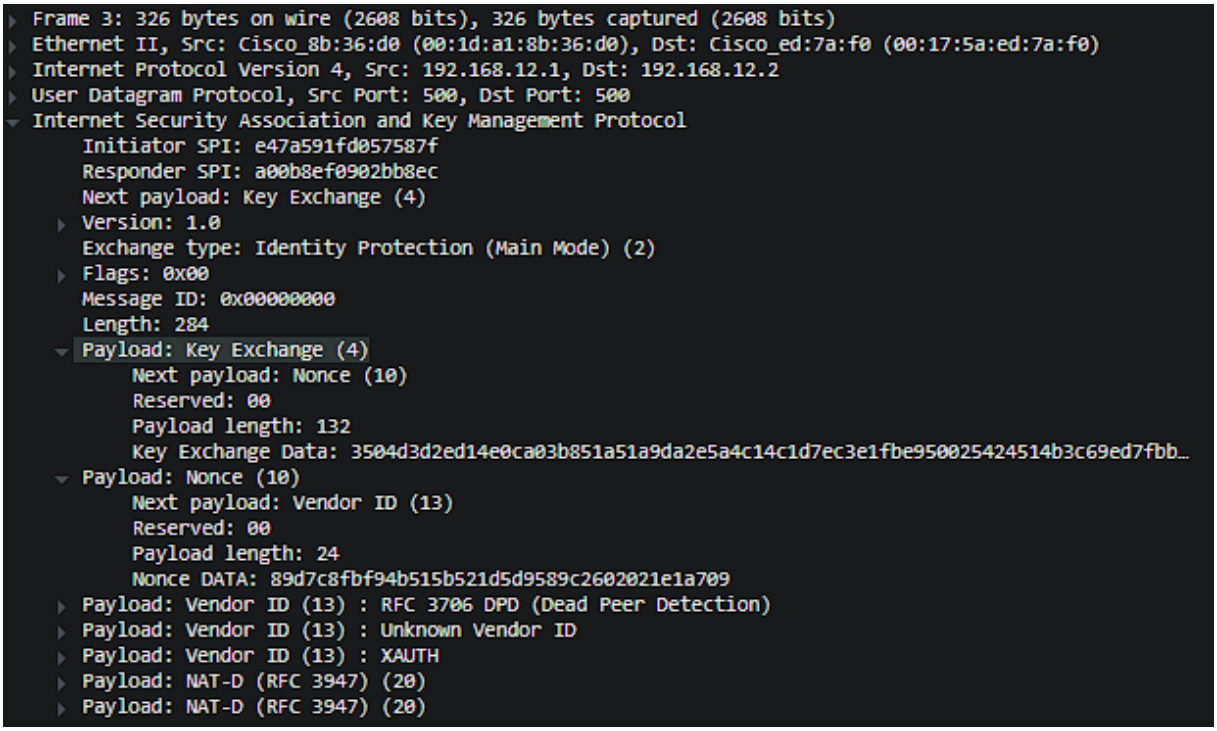
Sam IKEv1 protokol se sastoji od dvije faze. U prvoj fazi klijenti dogovaraju koje će protokole i algoritme koristiti za autentikaciju, enkripciju te razmjenu ključeva. Prva faza se može izvoditi u dva načina rada, „Main Mode“ i „Aggressive Mode“. „Main Mode“ se sastoji od ukupno 6 poruka dok se „Aggressive Mode“ sastoji od 3. Druga faza se sastoji od ukupno 3 poruke i koristi „Quick Mode“. U ovom radu će se za primjer uzeti „Main Mode“. Ako se klijenti dogovore da će za razmjenu ključeva koristiti Diffie-Hellman algoritam tada će se u drugom koraku taj algoritam izvršiti. Rezultat tog koraka će biti da oba klijenta imaju dogovoren dijeljeni ključ. Zadnji korak je autentikacija koja će biti izvršena dogovorenom metodom iz prvog koraka [8].

Sljedeća slika predstavlja prvi korak prve faze, odnosno jedan klijent inicira komunikaciju s drugim. Na slici vidimo da se koristi IPsec protokol (ISAKMP u Wiresharku) te po IKE atributima vidimo algoritme i protokole koji će se za daljnju komunikaciju koristiti [8].



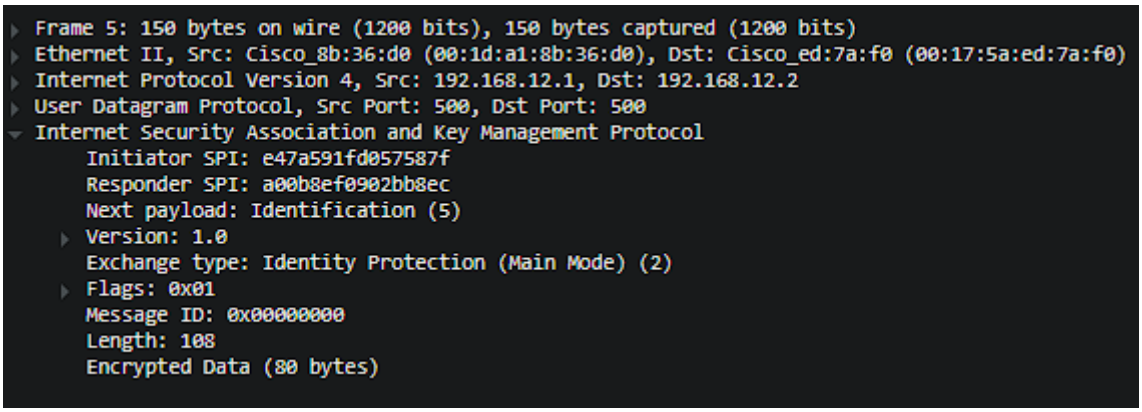
Slika 12: Prvi korak kod uspostave IPsec (Wireshark) [7]

Primatelj će tada potvrditi algoritme i protokole korištenja svojom porukom pošiljatelju. Na slici se može vidjeti kako nakon primitka odgovora primatelja odvija se razmjena ključa pomoću Diffie-Hellman algoritma. Šalje se nonce u obje strane te se računa dijeljeni ključ [8].



Slika 13: Diffie-Hellman u IPsec protokolu (Wireshark) [7]

Zadnje dvije poruke koje se šalju su kriptirane zbog toga jer su klijenti međusobno razmijenili ključeve te mogu slati kriptirane poruke. Njima se obavlja identifikacija i autentikacija svakog klijenta [8].



Slika 14: Kriptirana poruka nakon razmjene ključeva (Wireshark) [7]

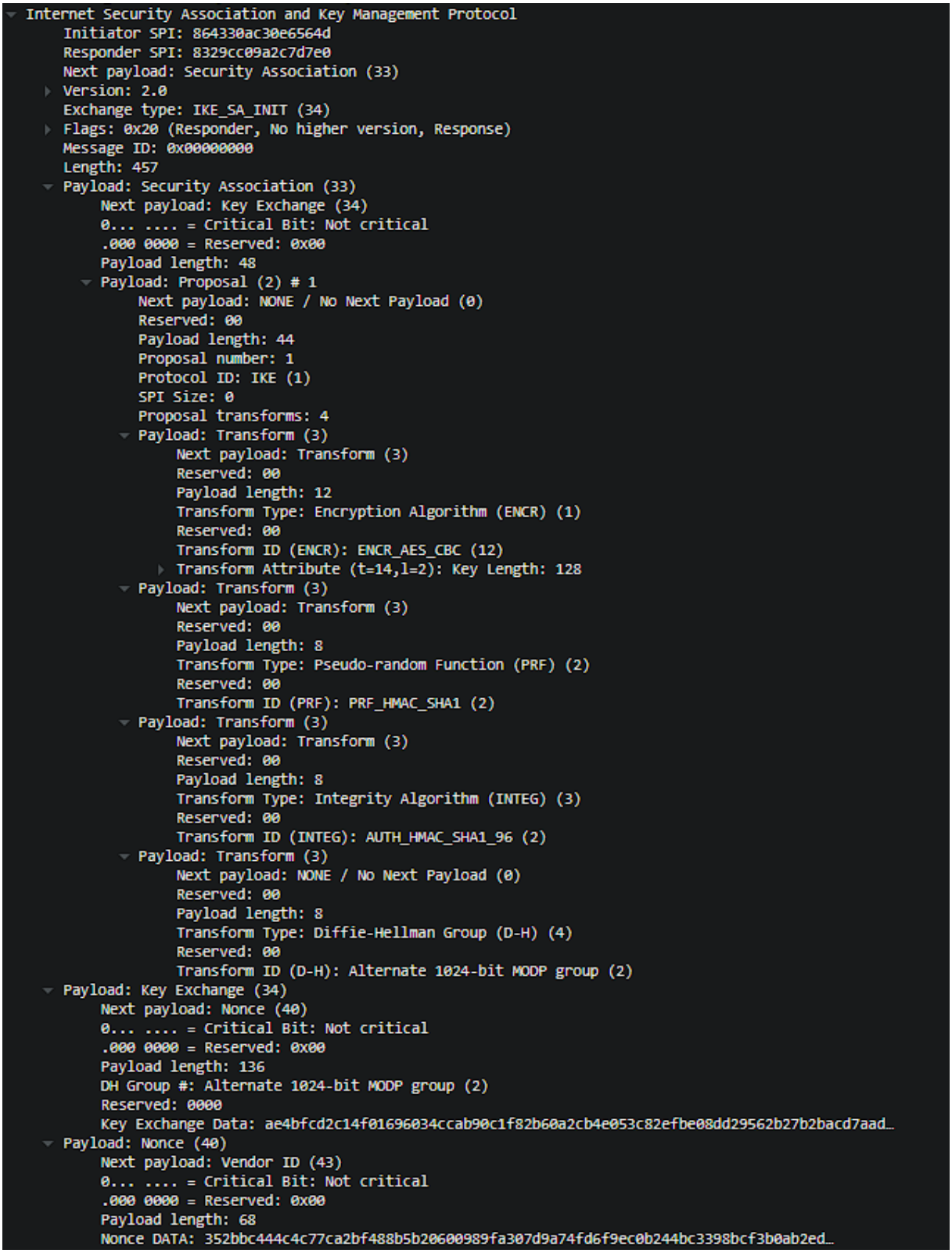
U drugoj IKE fazi se zapravo zaštićuju podaci. Isto kao i u prvoj fazi, klijenti se međusobno dogovaraju o protokolima i algoritmima koje će koristiti. IKEv1 u drugoj fazi koristi isti razmijenjeni tajni ključ no IKEv2 podržava PFS (Perfect Forward Secrecy) što znači da će u drugoj fazi koristiti drugačije tajne ključeve. [9]

5.2.2. IKEv2

IKEv2 također kao i IKEv1 ima dvije faze, no razlika je u tome da se obje faze izvode u 4 poruke umjesto 6 ili 9 kao IKEv1. Prva faza se sastoji od poruka „IKE_SA_INIT“ i „IKE_AUTH“. Prvi korak prve faze se zapravo sastoji od zahtjeva i odgovora koji oboje predstavljaju poruke „IKE_SA_INIT“. To je zapravo uspostava SA (Security Association) skupa protokola kojim se zapravo dogovaraju parametri kao kriptografski algoritmi i ostali protokoli ili algoritmi koji će se koristiti. U toj poruci se isto tako dogovara razmjena ključa odnosno može se birati koja će se varijacija Diffie-Hellman protokola koristiti te se šalje „nonce“. [10]

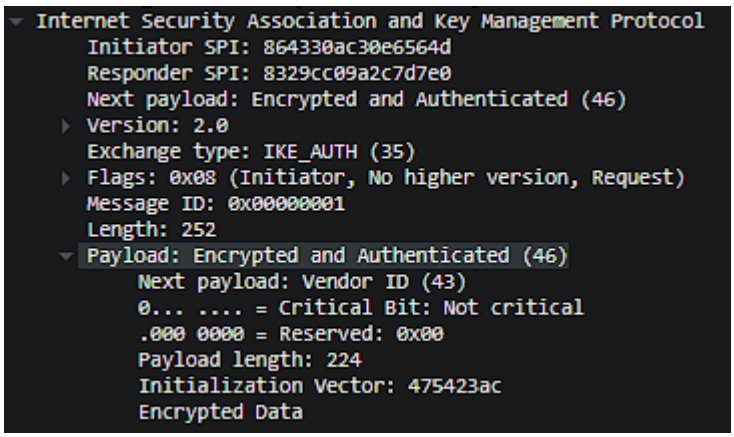
Dijeljeni ključevi koji će se koristiti za drugu fazu su ključevi za autentikaciju IKE sudionika komunikacije, autentikaciju poruka, enkripciju poruka te Diffie-Hellman grupa koja će se koristiti za izvođenje ključeva. [10]

Na slici ispod IKEv2 u radu može se vidjeti u Wiresharku. Red „Version: 2.0“ označuje da se koristi IKEv2 umjesto IKEv1. Kod „Exchange type:“ može se vidjeti da je na redu poruka „IKE_SA_INIT“. Ispod tog reda nalazi se i payload u kojem se može vidjeti da je korišten SA i svi algoritmi koji su dogovoreni za komunikaciju. Jedan od tih algoritama je jedna od grupa Diffie-Hellman protokola koji će se koristiti za razmjenu tajnog ključa. [10]



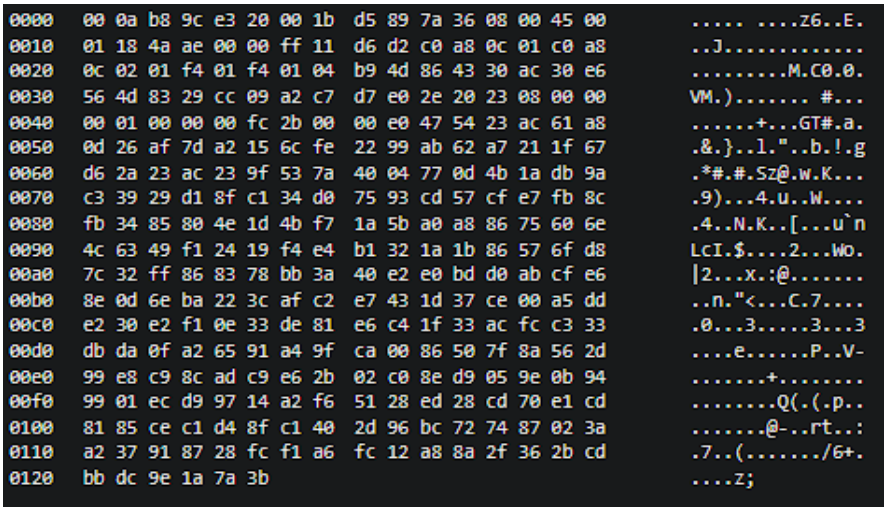
Slika 15: IKE_SA_INIT (Wireshark) [7]

Autentikacija se primatelja poruka se vrši porukama „IKE_AUTH“. Poruke koje su do sad spomenute nisu enkriptirane no sa porukom „IKE_AUTH“ enkripcija je prisutna. To se može i uočiti na sljedećoj slici koja prikazuje da je payload enkriptiran i autenticiran. [10]



Slika 16: IKE_AUTH (Wireshark) [7]

Sam payload nam trenutno ne predstavlja ništa te izgleda kako je prikazano na sljedećoj slici:



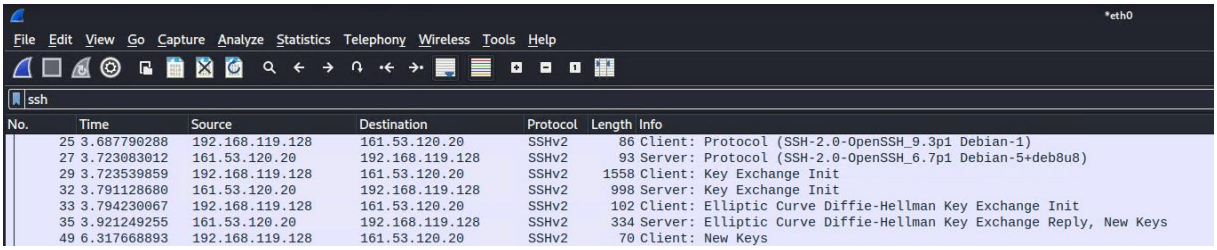
Slika 17: IKE_AUTH payload (Wireshark) [7]

5.3. SSH

Secure Shell (SSH) je protokol transportnog sloja OSI-ja za osiguravanje veze između klijenta i poslužitelja. Najčešće se koristi u slučaju kada administrator sustava želi sigurno pristupiti računalu preko nezaštićene mreže. To radi kako bi na daljinu mogao izvršavati naredbe te premještati datoteke s jednog računala na drugo. SSH protokol bazira se na asimetričnom kriptosustavu i sluša na portu 22 [11]. SSH handshake se sastoji od sljedećih koraka:

1. Razmjena verzija SSH protokola
2. Razmjena ključeva
3. Elliptic Curve Diffie-Hellman inicijalizacija
4. Elliptic Curve Diffie-Hellman odgovor
5. Novi ključevi

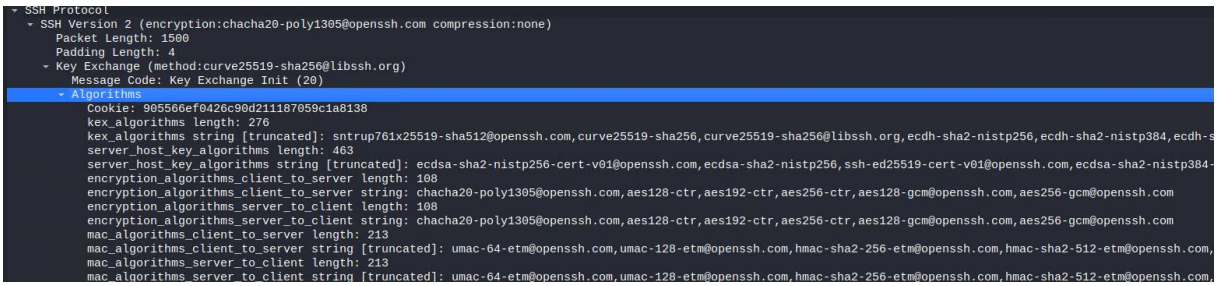
Navedeni koraci mogu se vidjeti na slici ispod.



Slika 18: Prikaz koraka kod uspostave SSH-a (Wireshark)

U prvom koraku klijent i server šalju niz verzija. To uočavamo na slici u prve dvije poruke, oboje koriste verziju SSHv2 i OpenSSH softver. Drugi korak označava razmjenu ključeva između klijenta i servera. Ona započinje tako da obje strane međusobno šalju poruku „SSH_MSH_KEX_INIT“ koja sadržava popis kriptografskih algoritama koje određena strana podržava. Redoslijed liste odnosno redoslijed zapisanih algoritama označava preferencije korištenja istih. Te liste se šalju istovremeno pa tako server i klijent svojom listom „pogađaju“ algoritme koji će se koristiti. Ako se ti algoritmi pogode prelazi se na sljedeći korak, a u

suprotnom se odabiru algoritmi koji se pogađaju a istovremeno imaju najveći mogući prioritet. Ukoliko obje strane ne mogu doći do sporazuma o algoritmu koji će se koristiti veza se obustavlja. Slika ispod predstavlja listu algoritama koju je klijent poslao serveru [11],[12].

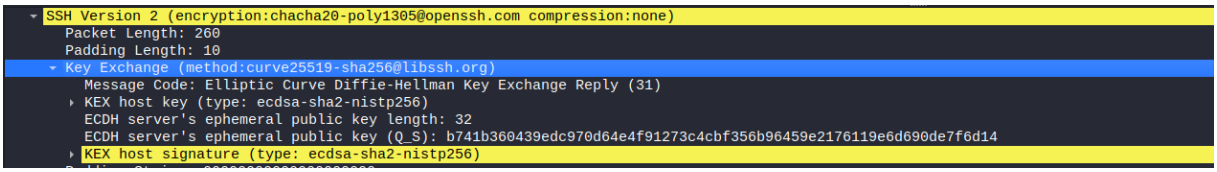


Slika 19: Lista mogućih algoritama koju klijent šalje serveru (Wireshark)

Inicijalizacija Elliptic Curve Diffie-Hellman-a predstavlja treći korak. Ta implementacija Diffie-Hellman-a označava da su ključne vrijednosti koje se koriste za input u algoritam neke odabrane točke na elipsi. Kada klijent generira par ključeva (javni i privatni ključ) tada ih šalje u poruci „SSH_MSG_KEX_ECDH_INIT“. Taj par ključeva je kratkotrajan jer će se koristiti samo u ovom slučaju. Tako je napadačima vrlo teško ukrasti nešto što jednostavno više ne postoji (forward secrecy) [11],[12].

Slika 20: Elliptic Curve Diffie-Hellman kod SSH-a (Wireshark)

Nakon što server primi javni ključ, generira svoj par ključeva te koristi sve poznate ključeve kako bi kreirao novi tajni ključ. Koristi algoritam sažimanja koji je odabran u koraku odabira algoritama na početku te enkriptira rezultat privatnim ključem. Poruku tada klijent može dekriptirati javnim ključem servera. Sastavljena poruka koju server šalje klijentu naziva se „SSH_MSG_KEX_ECDH_REPLY“ [11],[12].



Slika 21: „SSH_MSG_KEX_ECDH_REPLY“ poruka (Wireshark)

Kada klijent primi poruku, ima sve podatke koje treba kako bi izračunao tajni ključ i exchange hash. Ako ste ikada vidjeli poruku koja je prikazana na slici ispod, ona označava da prikazani ključ nije u vašoj lokalnoj bazi podataka poznatih hostova [11],[12].

```
The authenticity of host 10.10.10.10 (10.10.10.10)' can't be established.
ECDSA key fingerprint is SHA256:pnPn3SxExHtVGNdzbV0cRzUrtNhqZv+Pwdq/qGQPZO3.
Are you sure you want to continue connecting (yes/no)?
```

Slika 22: Nastavak povezivanja bez autentikacije

Time klijent može prihvatiti konekciju bez autentikacije, ukoliko vjeruje toj konekciji. Klijent na kraju može provjeriti rezultat funkcije sažimanja koju je on izračunao s rezultatom koji mu je server poslao u poruci „SSH_MSG_KEX_ECDH_REPLY“. Ukoliko su rezultati jednaki tada je server autenticiran [11],[12]. Posljednja stvar koja se još mora odraditi prije nego li započnemo s „bulk“ šifriranjem podataka, obje strane moraju generirati 6 ključeva:

- 2 ključa za šifriranje
- 2 vektora inicijalizacije (IV)
- 2 vektora za integritet

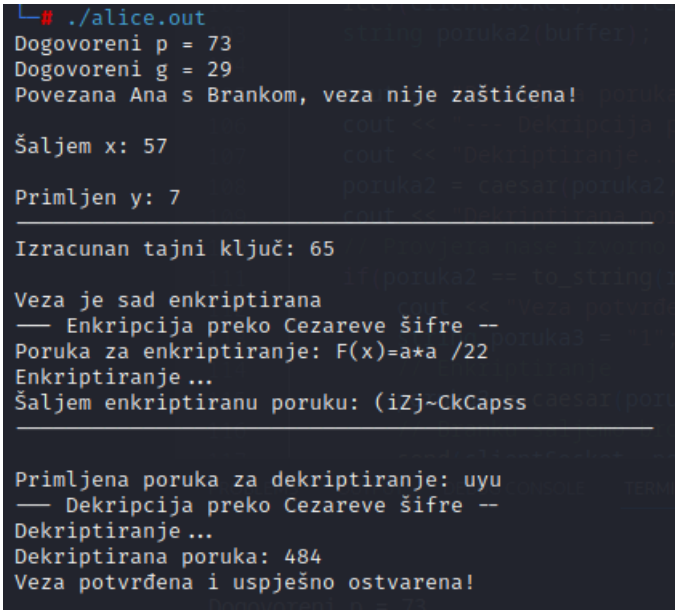
Ključevi se generiraju u parovima kako napadač ne bi mogao ponovno reproducirati zapis. Ključevi za šifriranje podataka koriste se sa simetričnom šifrom kako bi osigurali povjerljivost podataka odnosno za šifriranje i dešifriranje istih. Ključevi integriteta obično se koriste s kodom za provjeru autentičnosti (MAC) tako da se osiguramo da napadač ne manipulira šifriranim tekstom. Inicijalizacijski vektori obično su nasumični brojevi koji se koriste kao ulaz u simetričnu šifru. Oni se koriste da svaka poruka, iako je ista kao i neka druga, rezultira različitim šifriranim tekstom [11],[12].

6. Praktična komponenta

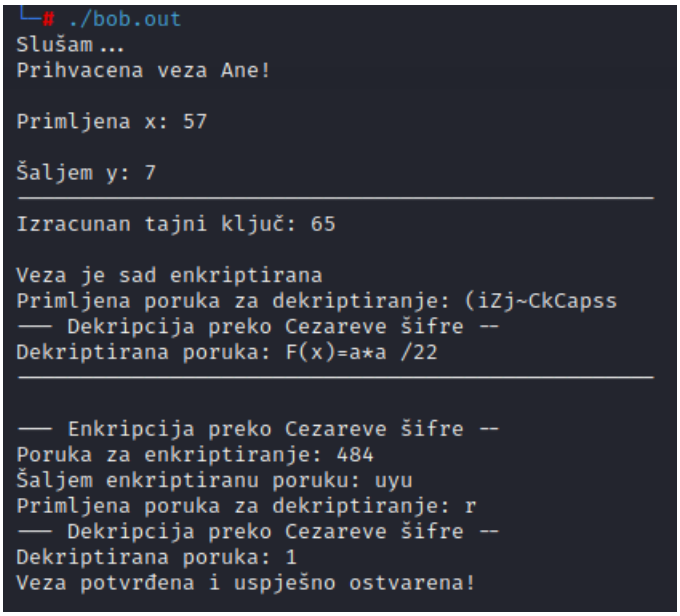
U praktičnoj komponenti prikazat će se par implementacija programskih rješenja sigurnosnih protokola za razmjenu ključeva kod ostvarenja veze između dva uređaja. Sve implementacije su izrađene u programskom jeziku C++ s uređivačem teksta Visual Studio Code. Kod svakog pojedinog protokola, poruke će se slati kao TCP pakete koji isprva nisu zaštićeni ali naš cilj je razmijeniti tajni ključ za enkripciju kako bismo tu nepogodu uklonili. Slikama, grafički će se prikazati kako se odvija proces razmjene ključeva korak po korak te ćemo opisati bitne dijelove. Osim toga, isječcima programskog koda objasnit će se kako je svako od tih programskih rješenja ostvareno i praktičnu komponentu ćemo završiti nekim zaključcima do kojih smo došli tijekom obrade.

6.1. Implementacija Diffie-Hellman sigurnosnog protokola

Teorija Diffie-Hellman protokola pokrivena je u jednom od prijašnjeg poglavlja, te je sad treba testirati u praktičnom primjeru da dokažemo kako se zapravo taj protokol ostvaruje u stvarnom svijetu i funkcionira kako je zamišljeno.



Slika 23: Rezultat programskog rješenja Ane



Slika 24: Rezultat programskog rješenja Branka

6.1.1. Upostava klijenta i servera

Želi li se uspostaviti komunikacija između dvaju ili više programa odnosno klijenta i servera, mora se kreirati socket preko kojeg će se međusobno povezati. To se izvodi funkcijom socket() te ona vraća broj kreiranog socketa koji zapisujemo u korisničku varijablu clientSocket kako bi znali poruke slati i primati na točno taj socket. Postavlja mu se IP na koji će se povezati te port na kojem server sluša. Nakon što su se podesili parametri adrese, funkcijom connect povezuje se na server koji već u međuvremenu sluša za nove konekcije. Funkciji se prosljeđuje clientSocket koji predstavlja broj socketa tipa integer na kojem se klijent spaja sa serverom. Nakon što je veza sa serverom uspostavljena, komunikacija između klijenta i servera može početi.

```
    // Parametri za stvaranje klijenta socketa i ostvarivanje veze s
Brankom
    int serverSocket, clientSocket;
    struct sockaddr_in serverAddress, clientAddress;
    socklen_t clientLength;

    clientSocket = socket(AF_INET, SOCK_STREAM, 0);

    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = inet_addr("127.0.0.1");
```

```
    serverAddress.sin_port = htons(8889);


    // Veza s Brankom je ostvarena, no sve poruke se trenutno salju u
citljivom obliku
    if(connect(clientSocket, reinterpret_cast<struct sockaddr
*>(&serverAddress), sizeof(serverAddress)) == 0){
        cout << "Povezana Ana s Brankom, veza nije zaštićena!" << endl <<
endl;
    }else{
        cerr << "Pogreska kod povezivanja s Brankom!" << endl;
        exit(1);
    }
```

Povezivanje klijenta na server ne bi bilo moguće ako server aktivno ne sluša za nadolazeće veze. Na slici ispod može se vidjeti isječak izvornog koda koji prikazuje kako se kreira socket za servera. Varijable serverAddress i clientAddress su po tipu strukture jer imaju više atributa po kojima se asociraju. Na isti način se postavljaju podaci adrese kao i kod klijenta samo u ovom scenariju se ne postavljaju IP i port na kojeg se želimo povezati već onaj na kojem će server slušati za nadolazeće veze. Funkcija bind povezuje lokalni naziv na socket. Nakon što se pozove funkcija socket(), taj socket zapravo nema ime pomoću kojeg ga se može asocirati. Slušanje za nove konekcije obavlja se funkcijom listen kojoj se prosljeđuje socket servera te maksimalni broj konekcija koje mogu biti aktivne (engl. queue). Kada se klijent poveže na server, funkcijom accept server prihvaća vezu te kreira novi socket te se upisuje u varijablu clientSocket. Nakon toga možemo komunicirati s klijentom preko varijable clientSocket.

```
// Parametri za stvaranje socketa servera i ostvarivanje veze s Brankom
    int serverSocket, clientSocket;
    sockaddr_in serverAddr, clientAddr;
    socklen_t clientAddrLen = sizeof(clientAddr);

    // Kreiranje socketa
    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1) {
        cerr << "Pogreska kod stvaranja socketa!" << endl;
        exit(1);
    }
```

```
    // Upisivanje svih bitnih dijelova serverAddr objekta kako bi rekli
serveru gdje da slusa i koje veze da prima
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(8889);


    // Bindanje socketa
    if (bind(serverSocket, (struct sockaddr*)&serverAddr,
sizeof(serverAddr)) == -1) {
        cerr << "Pogreska kod bindanja socketa!" << endl;
        close(serverSocket);
        exit(1);
    }


    // Server slusa za nadolazece veze, queue velicine 1
    if (listen(serverSocket, 1) == -1) {
        cerr << "Pogreska kod slusanja za veze!" << endl;
        close(serverSocket);
        exit(1);
    }else{
        cout << "Slušam..." << endl;
    }


    // Server prihvaca vezu od klijenta
    clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddr,
&clientAddrLen);
    if(clientSocket < 0){
        cerr << "Pogreska kod prihvacanja klijenta!" << endl;
        exit(1);
    }else{
        cout << "Prihvacena veza Ane!" << endl <<endl;
    }
```

6.1.2. Simetričan algoritam Cezarove šifre

Implementacija ovog protokola bazira se na simetričnom kriptosustavu te je iz tog razloga primijenjen jednostavan algoritam Cezarove šifre jer sam algoritam enkripcije ne mijenja smisao Needham-Schroeder protokola već samo jačinu sigurnosti.

```
string caesar(string text, int s, int encDec)
{
    string result = "";

    s = s % 95;
    if(encDec == 1){
        for (int i = 0; i < text.length(); i++) {
            if (int(text[i]) > 31 && int(text[i]) < 127)
                result += char(int(text[i] + s - 32) % 95 + 32);
        }
    }else if(encDec == 0){
        for (int i = 0; i < text.length(); i++) {
            if (int(text[i]) > 31 && int(text[i]) < 127)
            {
                int c = char(int(text[i] - s));
                if(c < ' '){
                    c = c + '~' - ' ' + 1;
                }
                result += c;
            }
        }
    }

    return result;
}
```

Simetričan algoritam Cezarove šifre te dio za implementaciju uspostave jednostavne TCP veze između klijenta i servera neće se prikazivati u daljnjim implementacijama jer su već prikazane kod implementacije Diffie-Hellman protokola.

6.1.3. Funkcija splitString

Funkcija splitString je korisnička funkcija koja je u ovim implementacijama napravljena da služi za razdvajanje primljenih poruka kako bismo došli do svakog zasebnog dijela poruke posebno. Neke dijelove poruka će se koristiti za daljnje računanje te postavljanje drugih varijabli pa nam je iz tog razloga potrebna.

```
vector<string> splitString(const string& stringForSplit, char delimiter)
{
    vector<string> parts;
    istringstream iss(stringForSplit);
    string part;

    while (getline(iss, part, delimiter))
    {
        parts.push_back(part);
    }

    return parts;
}
```

6.1.4. Programsko rješenje Ana

Programski kod ispod prikazuje i opisuje dio koji Ana obavlja kako je i opisano u Diffie-Hellman poglavlju.

Funkcija send označava slanje poruke te joj se za argumente prosljeđuju broj socketa preko kojeg šalje poruku, sama poruka te duljina poruke. Zadnji broj 0 predstavlja zastavice koje se u ovom primjeru ne koriste. Buffer označava statično polje tipa char u koje će se spremati sve nadolazeće poruke. Funkcija recv označava slušanje na određenom socketu za nadolazeće poruke na već uspostavljenoj vezi. Program na toj liniji staje s izvršavanjem sve dok ne dobije odgovor. Nakon što odgovor stigne, ispisuje se u šifriranom obliku te se funkcijom caesar dekriptira kako bi se vidio i ispisao sadržaj dekriptirane poruke.

```
    // Racunamo x te ga saljemo Branku
    long long int a = 4;
    long long int x = (long long int)pow(g, a) % p;
    cout << "Šaljem x: " << x << endl << endl;
    send(clientSocket, to_string(x).c_str(), to_string(x).length(), 0);

    // Primamo y od Branka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string y(buffer);
    cout << "Primljen y: " << y << endl;
    cout << "-------------------------------------------------" << endl;

    // Racunamo tajni kljuc k koji cemo koristiti za simetricnu enkripciju
poruka
    long long int k = (long long int)pow(stoi(y), a) % p;
    cout << "Izracunan tajni ključ: " << k << endl << endl;
    cout << "Veza je sad enkriptirana" << endl;



    int testA = 22;
    int rezultat = testA*testA;
    string poruka = "F(x)=a*a /" + to_string(testA);
    string porukaEnc = poruka;
    cout << "--- Enkripcija preko Cezareve šifre --" << endl;
    cout << "Poruka za enkriptiranje: " << poruka << endl;
    cout << "Enkriptiranje..." << endl;
    porukaEnc = caesar(poruka, k, 1);
    cout << "Šaljem enkriptiranu poruku: " << porukaEnc << endl;
    cout << "-------------------------------------------------" << endl <<
endl;
    // Saljemo testnu poruku kako bi provjerili da je enkriptirana veza
ostvarena
    // te da oba klijenta mogu enkriptirati i dekriptirati poruke
    send(clientSocket, porukaEnc.c_str(), porukaEnc.length(), 0);

    // Primamo istu poruku natrag kako bi utvrdili da je to poruka od
Branka te da on ima tajni kljuc
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
```

```
    string poruka2(buffer);

    cout << "Primljena poruka za dekriptiranje: " << poruka2 << endl;
    cout << "--- Dekripcija preko Cezareve šifre --" << endl;
    cout << "Dekriptiranje..." << endl;
    poruka2 = caesar(poruka2, k, 0);
    cout << "Dekriptirana poruka: " << poruka2 << endl;
    // Provjera nase izvorno poslane porue i poruke primljene od branka
    if(poruka2 == to_string(rezultat)){
        cout << "Veza potvrđena i uspješno ostvarena!" << endl;
        string poruka3 = "1";
        // Enkriptiranje
        poruka3 = caesar(poruka3, k, 1);
        // Branku saljemo broj 1 odnosno potvrdu da je konekcija ostvarena
        send(clientSocket, poruka3.c_str(), poruka3.length(), 0);
    }else{
        cerr << "Nonce nije jednak poslanom!" << endl;
        exit(1);
    }

    // Zatvaramo socket kada smo zavrsili sa komunikacijom
    close(clientSocket);
```

6.1.5. Programsko rješenje Branko

Programski kod ispod prikazuje i opisuje dio koji Branko obavlja kako je i opisano u Diffie-Hellman poglavlju.

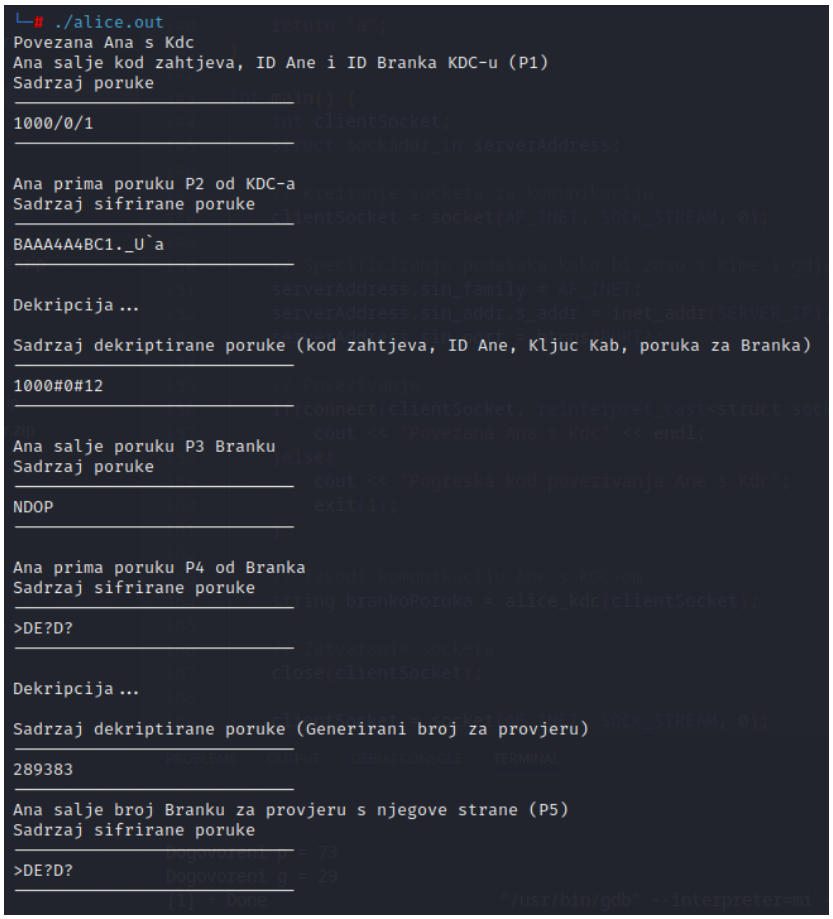
```
// Branko prima x od Ane
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string x(buffer);
    cout << "Primljena x: " << x << endl << endl;

    int b = 3;
    // Racunamo y te ga saljemo Ani
    int y = (long long int)pow(g, b) % p;
```

```
    cout << "Šaljem y: " << y << endl;
    cout << "-------------------------------------------------" << endl;
    send(clientSocket, to_string(y).c_str(), to_string(y).length(), 0);

    // Racunamo k odnosno tajni kljuc koji cemo koristiti za enkripciju i
dekripciju daljnjih poruka
    int k = (long long int)pow(stoi(x),b) % p;
    cout << "Izracunan tajni ključ: " << k << endl << endl;
    cout << "Veza je sad enkriptirana" << endl;

    // Branko prima testnu poruku od Ane kako bi potvrdili da oboje imaju
isti kljuc za komunikaciju
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string poruka(buffer);

    cout << "Primljena poruka za dekriptiranje: " << poruka << endl;
    cout << "--- Dekripcija preko Cezareve šifre --" << endl;
    poruka = caesar(poruka, k, 0);
    cout << "Dekriptirana poruka: " << poruka << endl;

    // Branko dijeli poruku na 3 dijela kako bi vidio koja te funkcija
(f(x)) kojom racuna rezultat
    // te koji je x
    vector<string> parts = splitString(poruka, '/');

    // Branko racuna rezultat f(x)
    int num = stoi(parts[1]) * stoi(parts[1]);

    string poruka2 = caesar(to_string(num), k, 1);
    cout << "-------------------------------------------------" << endl <<
endl;
    cout << "--- Enkripcija preko Cezareve šifre --" << endl;
    cout << "Poruka za enkriptiranje: " << to_string(num) << endl;
    cout << "Šaljem enkriptiranu poruku: " << poruka2 << endl;
    // Branko salje izracunati f(x)
    send(clientSocket, poruka2.c_str(), poruka2.length(), 0);

    //Branko prima potvrdu da je i Ana potvrdila konekciju
    memset(buffer, 0, BUFFER_SIZE);
```

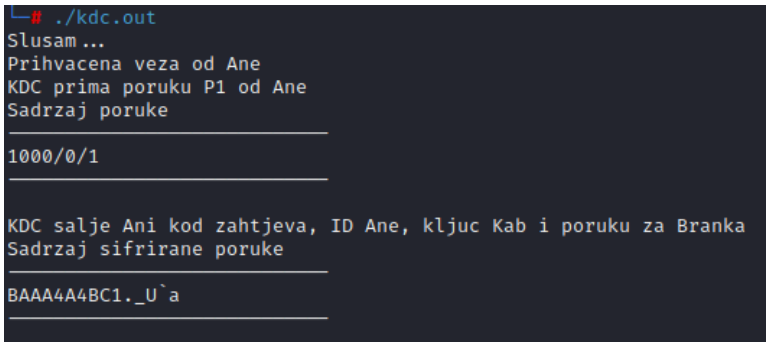
```
recv(clientSocket, buffer, BUFFER_SIZE, 0);
string poruka3(buffer);

cout << "Primljena poruka za dekriptiranje: " << poruka3 << endl;
cout << "--- Dekripcija preko Cezareve šifre --" << endl;
poruka3 = caesar(poruka3, k, 0);
cout << "Dekriptirana poruka: " << poruka3 << endl;
if(stoi(poruka3) == 1){
    cout << "Veza potvrđena i uspješno ostvarena!" << endl;
}else{
    cerr << "Pogreška kod povezivanja" << endl;
    exit(1);
}
// Zatvaramo socket kada smo zavrsili sa komunikacijom
close(clientSocket);
```

6.2. Implementacija Needham-Schroeder protokola

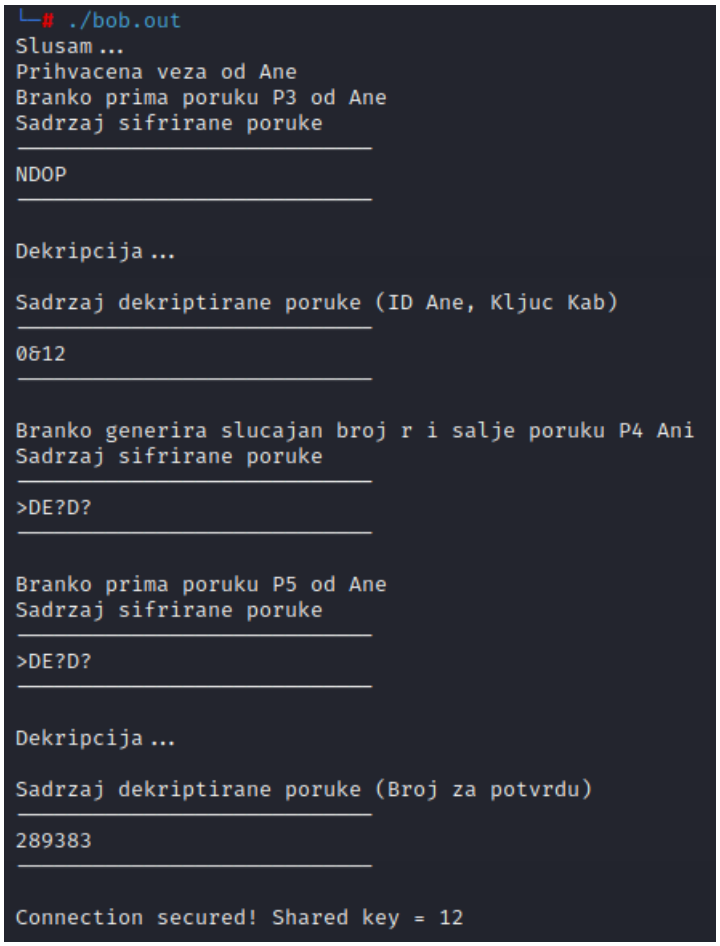
Kako je sam koncept i teorija Needham-Schroeder protokola obrađena u jednom od prijašnjih poglavlja, sada će se taj protokol prikazati na djelu pomoću programskog rješenja. Sljedeće 3 slike prikazuju 3 pokrenuta programa koji zapravo funkcioniraju kao krajnje točke u komunikaciji koje su već spomenute (Ana, Branko, KDC). KDC predstavlja server koji sluša na određenom portu i kada se Ana poveže na njega kreće izvršavanje protokola. Isto tako, Branko sluša kako bi mogao primiti konekciju od Ane nakon što ona dobije poruku od KDC-a da Branku može proslijediti ključ s kojim će komunicirati.



Slika 25: Rezultat uspostave veze (Ana)



Slika 26: Rezultat uspostave veze (KDC)



Slika 27: Rezultat uspostave veze (Branko)

6.2.1. Programsko rješenje Ana

Sljedeći isječak iz izvornog koda je funkcija kojom Ana komunicira s KDC-om. Sastavlja se poruka i šalje u jasnom obliku iz razloga što još uvijek nije uspostavljena enkriptirana veza. Funkcija send označava slanje poruke te joj se za argumente prosljeđuju broj socketa preko kojeg šalje poruku, sama poruka te duljina poruke. Zadnji broj 0 predstavlja zastavice koje se u ovom primjeru ne koriste. Na sličan način funkcioniraju sva daljnja slanja u sva tri programska rješenja pa iz tog razloga nije potrebno svako zasebno objasniti.

Buffer označava statično polje tipa char u koje će se spremati sve nadolazeće poruke. Funkcija recv označava slušanje na određenom socketu za nadolazeće poruke na već

uspostavljenoj vezi. Program na toj liniji staje s izvršavanjem sve dok ne dobije odgovor. Nakon što odgovor stigne, ispisuje se u šifriranom obliku te se funkcijom caesar dekriptira kako bi se vidio i ispisao sadržaj dekriptirane poruke. Kao i slanja, primanja su implementirana na sličan način u sva tri programska rješenja te iz tog razloga se neće objašnjavati svako zasebno.

```
string alice_kdc(int clientSocket) {
    // Slanje poruke P1 po dijagramu toka
    string p1 = "1000/0/1";
    cout << "Ana salje kod zahtjeva, ID Ane i ID Branka KDC-u (P1)" <<
endl;
    cout << "Sadrzaj poruke" << endl;
    cout << "----------------------------" << endl;
    cout << p1 << endl;
    cout << "----------------------------" << endl << endl;
    send(clientSocket, p1.c_str(), p1.length(), 0);

    // Ana prima poruku P2 po dijagramu toka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string p2(buffer);

    cout << "Ana prima poruku P2 od KDC-a" << endl;
    cout << "Sadrzaj sifrirane poruke" << endl;
    cout << "----------------------------" << endl;
    cout << p2 << endl;
    cout << "----------------------------" << endl << endl;

    // Ana svojim kljucem Ka dekriptira poruku response i dobije Ra,IDa,Kab
te saznaje kljuc Kab
    cout << "Dekripcija..." << endl << endl;
    p2 = caesar(p2, kljuc, 0);

    vector<string> parts = splitString(p2, '|');
    cout << "Sadrzaj dekriptirane poruke (kod zahtjeva, ID Ane, Kljuc Kab,
poruka za Branka)" << endl;
    cout << "----------------------------" << endl;
    cout << parts[0] << endl;
    cout << "----------------------------" << endl << endl;
```

```
    vector<string> parts2 = splitString(parts[0], '#');
    kljucKab = stoi(parts2[2]);

    //Razdvajamo poruku tako da branku saljemo samo c1
    string brankoPoruka = parts[1];
    return brankoPoruka;
}




void alice_bob(int clientSocket, string p3) {
    cout << "Ana salje poruku P3 Branku"<< endl;
    cout << "Sadrzaj poruke" << endl;
    cout << "----------------------------" << endl;
    cout << p3 << endl;
    cout << "----------------------------" << endl << endl;

    // Ana salje poruku P3 po dijagramu toka
    send(clientSocket, p3.c_str(), p3.length(), 0);

    // Ana prima poruku P4 po dijagramu toka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);

    string p4(buffer);
    cout << "Ana prima poruku P4 od Branka" << endl;
    cout << "Sadrzaj sifrirane poruke" << endl;
    cout << "----------------------------" << endl;
    cout << p4 << endl;
    cout << "----------------------------" << endl << endl;

    cout << "Dekripcija..." << endl << endl;
    p4 = caesar(p4, kljucKab, 0);

    cout << "Sadrzaj dekriptirane poruke (Generirani broj za provjeru)" <<
endl;
```

```
    cout << "----------------------------" << endl;
    cout << p4 << endl;
    cout << "----------------------------" << endl;

    string p5 = caesar(p4, kljucKab, 1);

    cout << "Ana salje broj Branku za provjeru s njegove strane (P5)" <<
endl;
    cout << "Sadrzaj sifrirane poruke" << endl;
    cout << "----------------------------" << endl;
    cout << p5 << endl;
    cout << "----------------------------" << endl << endl;
    // Ana salje poruku P5 po dijagramu toka
    send(clientSocket, p5.c_str(), p5.length(), 0);
}
```

6.2.2. Programsko rješenje Branko

Sve što je spomenuto u opisu kod programskog rješenja Ane vrijedi i za sljedeća rješenja. Programskim kodom ispod implementira se klijent Branko i ponaša se kako je to opisano u dijagramu kod opisa Needham-Schroeder protokola. Funkcija „alice_bob“ predstavlja komunikaciju Ane s Brankom u kojoj oni uspostavljaju vezu nakon što je Ani isporučen ključ K_{AB} koji je generirao KDC.

```
void alice_bob(int clientSocket) {
    // Branko prima poruku P3 po dijagramu toka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string p3(buffer);

    cout << "Branko prima poruku P3 od Ane" << endl;
    cout << "Sadrzaj sifrirane poruke" << endl;
    cout << "----------------------------" << endl;
    cout << p3 << endl;
    cout << "----------------------------" << endl << endl;

    cout << "Dekripcija..." << endl << endl;
```

```
    p3 = caesar(p3, kljuc, 0);

    cout << "Sadrzaj dekriptirane poruke (ID Ane, Kljuc Kab)" << endl;
    cout << "----------------------------" << endl;
    cout << p3 << endl;
    cout << "----------------------------" << endl << endl;

    vector<string> parts = splitString(p3, '&');
    kljucKab = stoi(parts[1]);
    // Branko generira slucajan broj r i salje poruku Ani s slucajnim brojem
r kriptiranu s kljucem Kab
    string r = to_string(rand() % 1000000);
    string p4 = caesar(r, kljucKab, 1);

    cout << "Branko generira slucajan broj r i salje poruku P4 Ani" << endl;
    cout << "Sadrzaj sifrirane poruke" << endl;
    cout << "----------------------------" << endl;
    cout << p4 << endl;
    cout << "----------------------------" << endl << endl;

    // Branko salje poruku P4 po dijagramu toka
    send(clientSocket, p4.c_str(), p4.length(), 0);

    // Branko prima poruku P5 po dijagramu toka
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string p5(buffer);

    cout << "Branko prima poruku P5 od Ane" << endl;
    cout << "Sadrzaj sifrirane poruke" << endl;
    cout << "----------------------------" << endl;
    cout << p5 << endl;
    cout << "----------------------------" << endl << endl;

    cout << "Dekripcija..." << endl << endl;
    p5 = caesar(p5, kljucKab, 0);

    cout << "Sadrzaj dekriptirane poruke (Broj za potvrdu)" << endl;
    cout << "----------------------------" << endl;
    cout << p5 << endl;
```

```
    cout << "----------------------------" << endl << endl;

    if(p5 == r){
        cout << "Konekcija ostvarena! Tajni ključ = " + to_string(kljucKab)
<< endl;
    }else{
        cerr << "Poslani nonce nije jednak primljenom!" << endl;
        exit(1);
    }

}
```

6.2.3. Programsko rješenje KDC

Programski kod ispod predstavlja centar za distribuciju ključeva koji će generirati tajni ključ K_{AB} i dodijeliti ga Ani i Branku kako bi oni međusobno mogli uspostaviti enkriptiranu komunikaciju. Funkcija „alice_kdc“ označava komunikaciju između Ane i KDC-a na način kako je to objašnjeno u poglavlju Needham-Schroeder.

```
void alice_kdc(int clientSocket) {
    // Primanje poruka P1 po dijagramu toka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string kljucKab = "12";
    string response(buffer);
    cout << "KDC prima poruku P1 od Ane" << endl;
    cout << "Sadrzaj poruke" << endl;
    cout << "----------------------------" << endl;
    cout << response << endl;
    cout << "----------------------------" << endl << endl;

    vector<string> parts = splitString(response, '/');

    // Sastavljanje poruke c1
    string c1 = parts[1] +"&" + kljucKab;
    c1 = caesar(c1, kljuceviKlijenta[1], 1);
```

```
    // Sastavljanje poruke m2
    string m2 = parts[0] + "#" + parts[1] + "#" + kljucKab + " |" + c1;
    m2 = caesar(m2, kljuceviKlijenta[0], 1);
    // Posalji poruku M2 po dijagramu toka

    cout << "KDC salje Ani kod zahtjeva, ID Ane, kljuc Kab i poruku za
Branka" << endl;
    cout << "Sadrzaj sifrirane poruke" << endl;
    cout << "----------------------------" << endl;
    cout << m2 << endl;
    cout << "----------------------------" << endl << endl;

    send(clientSocket, m2.c_str(), m2.length(), 0);
}
```

6.3. Implementacija raspodijeljene raspodjele tajnih ključeva

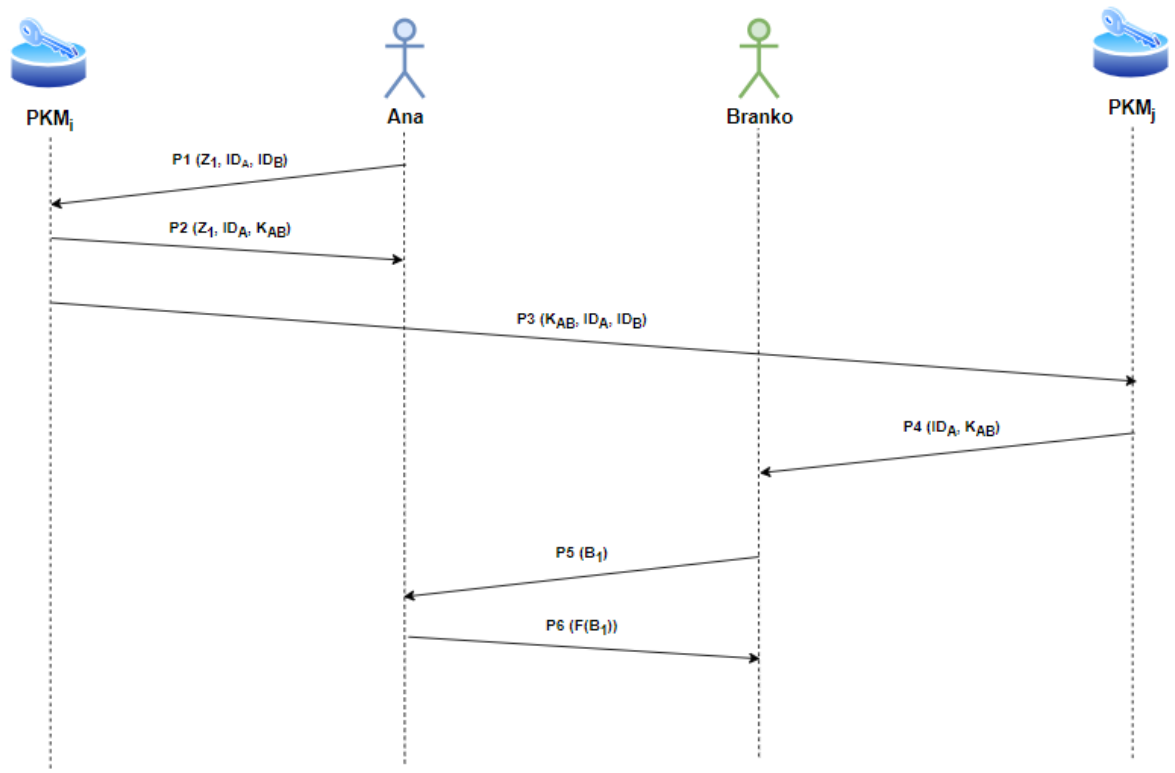
Ukoliko postoji samo jedan centar za raspodjelu ključeva, dolazimo do nedostataka kao što su preopterećenost sustava i smanjena pouzdanost. Taj problem može se riješiti tako da umjesto jednog centra, postoje n centrova koji imaju istu funkciju samo što je ukupni broj ključeva u jednom centru n puta manji. Sljedeći korak je opisati kako protokol funkcionira da se lakše razumije rješenje napravljeno programskim jezikom C++.

U prvom koraku Ana porukom P_1 šalje kod zahtjeva kojim će PKM znati što raditi, svoj identifikacijski broj te identifikacijski broj Branka. Nakon što je poruku zaprimio PKM_i, on će generirati ključ K_{AB}, poslati ga Ani (P_2). Isto tako, vidjet će da se Branko ne nalazi u njegovom popisu klijenata već je za njega zadužen PKM_j.

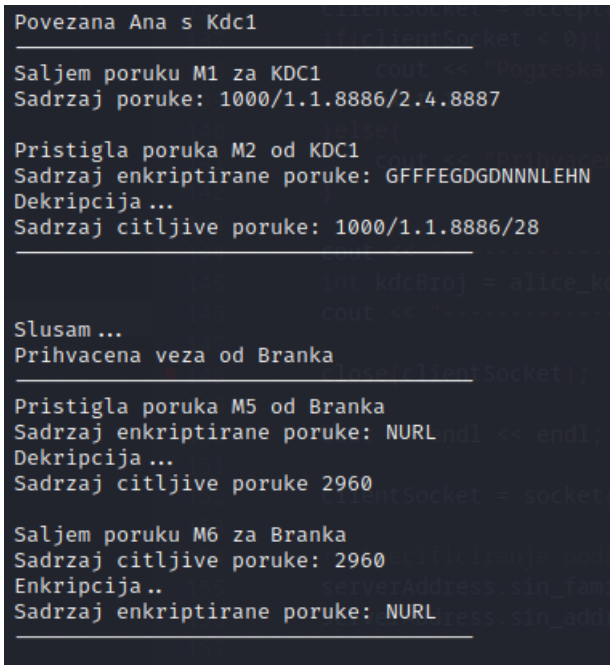
Kako bi mogao Branku isporučiti ključ K_{AB}, kontaktirat će PKM_j i u poruci će mu poslati generirani ključ K_{AB} te identifikacijski broj Ane ID_A i Branka ID_B (P_3). PKM_j će tada dekriptirati poruku i vidjeti da Branka mora obavijestiti o tome da Ana želi uspostaviti komunikaciju s njime. To radi tako da Branku šalje poruku P_4 u kojoj su identifikacijski broj Ane ID_A te zajednički ključ K_{AB}.

U trenutku kada Branko primi poruku P_4 vidjet će da Ana želi komunicirati s njime te će joj porukom P_5 poslati generirani broj B_1 kako bi je mogao autenticirati. Ana koristi broj

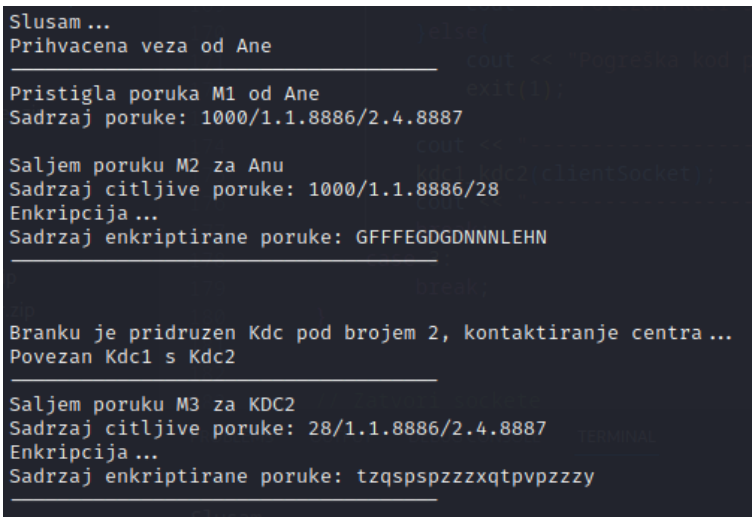
B_1 kako bi unaprijed dogovorenom funkcijom izračunala rezultat i poslala ga natrag Branku (P_6). Branko inverzom potvrđuje da je to uistinu Ana te prihvaća konekciju.



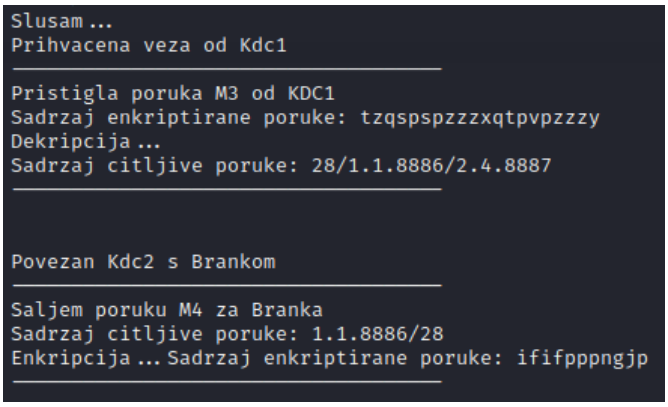
Slika 28: Tijek poruka protokola (Prema: Budin, Golub, Jakobović i Jelenković, 2016)



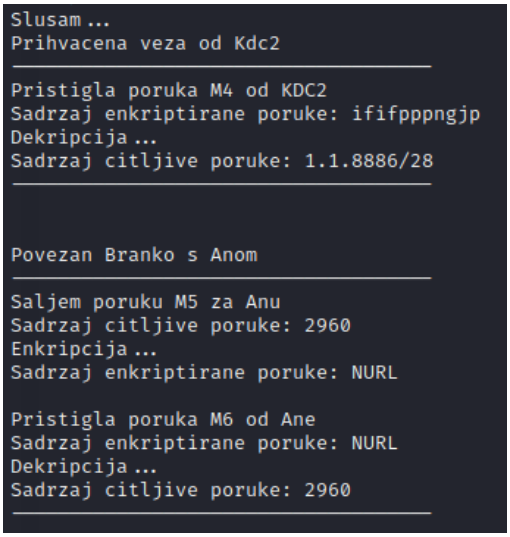
Slika 29: Rezultat uspostave veze (Ana)



Slika 30: Rezultat uspostave veze (KDC1)



Slika 31: Rezultat uspostave veze (KDC2)



Slika 32: Rezultat uspostave veze (Branko)

6.3.1. Programsko rješenje Ana

Anino programsko rješenje bazira se na dvije funkcije. Prva funkcija „alice_kdc1“ označava komunikaciju između Ane i KDC1 a druga Branka i Ane kako je to opisano u prošlom poglavlju.

```
void alice_kdc1(int clientSocket){
    // ID klijenta s kojim Ana zeli komunicirati
    idCommunication = "2.4.8887";
    string p1 = "1000/" + id + "/" + idCommunication;
    cout << "Saljem poruku p1 za KDC1" << endl << "Sadrzaj poruke: " << p1
<< endl << endl;
    // Slanje poruke P1 po dijagramu toka
```

```
    send(clientSocket, p1.c_str(), p1.length(), 0);


    // Primanje poruke P2 po dijagramu toka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string p2(buffer);
    cout << "Pristigla poruka p2 od KDC1" << endl << "Sadrzaj enkriptirane
poruke: " << p2 << endl << "Dekripcija..." << endl;
    // Dekriptiranje poruke
    string m2Decrypted = caesar(p2, key, 0);
    cout << "Sadrzaj citljive poruke: " << m2Decrypted << endl;
    vector<string> parts = splitString(m2Decrypted, '/');
    // Ana dobiva Kab te ga zapisuje u globalnu varijablu kako bi ga zapamtila
za daljnju komunikaciju
    kab = stoi(parts[2]);
}


void bob_alice(int clientSocket){
    // Primanje poruke P5 po dijagramu toka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string p5(buffer);
    // Dekriptiranje poruke
    string m5Decrypted = caesar(p5, kab, 0);
    cout << "Pristigla poruka p5 od Branka" << endl << "Sadrzaj enkriptirane
poruke: " << p5 << endl << "Dekripcija..." << endl;
    cout << "Sadrzaj citljive poruke " << m5Decrypted << endl << endl;


    cout << "Saljem poruku p6 za Branka" << endl << "Sadrzaj citljive poruke:
" << m5Decrypted << endl << "Enkripcija.." << endl;
    string m6Encrypted = caesar(m5Decrypted, kab, 1);
    cout << "Sadrzaj enkriptirane poruke: " << m6Encrypted << endl;
    // Slanje poruke P6 po dijagramu toka
    send(clientSocket, m6Encrypted.c_str(), m6Encrypted.length(), 0);
}
```

6.3.2. Programsko rješenje KDC1

Programsko rješenje KDC1 predstavlja KDC_i koji se sastoji od dvije funkcije („alice_kdc1“ i „kdc1_kdc2“). Prva funkcija temelji se na komunikaciji Ane i KDC1 kako bi Ana mogla zaprimiti ključ K_{AB}, a druga funkcija predstavlja komunikaciju između KDC1 i KDC2 (KDC_i i KDC_j).

```
int alice_kdc1(int clientSocket){
    // Primanje poruke P1 po dijagramu toka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string p1(buffer);
    cout << "Pristigla poruka p1 od Ane" << endl << "Sadrzaj poruke: " <<
p1 << endl << endl;

    vector<string> parts = splitString(p1, '/');
    // Zapisivanje ida i idb u globalne varijable
    ida = parts[1];
    idb = parts[2];

    // Provjera broja zahtjeva klijenta da KDC1 zna sto raditi
    if(stoi(parts[0]) == 1000){
        // Generiranje Kab kljuca za Anu i Branka
        kab = rand() % 100;
        string p2 = parts[0] + "/" + parts[1] + "/" + to_string(kab);
        vector<string> parts2 = splitString(parts[1], '.');
        cout << "Saljem poruku p2 za Anu" << endl << "Sadrzaj citljive
poruke: " << p2 << endl << "Enkripcija..." << endl;
        // Enkripcija
        string m2Encrypted = caesar(p2, clientKeys[stoi(parts2[1])], 1);
        cout << "Sadrzaj enkriptirane poruke: " << m2Encrypted << endl;
        // Slanje poruke P2 po dijagramu toka
        send(clientSocket, m2Encrypted.c_str(), m2Encrypted.length(), 0);
        vector<string> parts3 = splitString(parts[2], '.');
        // Vracamo broj KDC-a koji je podijeljen Branku, nalazi se u prvoj
sekciji ID-ja
        return stoi(parts3[0]);
    }else{
```

```
        cerr << "Kod zahtjeva je pogrešan!" << endl;
        exit(1);
    }
}

void kdc1_kdc2(int clientSocket){
    string p3 = to_string(kab) + "/" + ida + "/" + idb;
    cout << "Saljem poruku p3 za KDC2" << endl << "Sadrzaj citljive poruke:
" << p3 << endl << "Enkripcija..." << endl;
    // Enkriptiranje
    string m3Encrypted = caesar(p3, listaKdc[1].key, 1);
    cout << "Sadrzaj enkriptirane poruke: " << m3Encrypted << endl;
    // SLanje poruke P3 po dijagramu toka
    send(clientSocket, m3Encrypted.c_str(), m3Encrypted.length(), 0);
}
```

6.3.3. Programsko rješenje KDC2

KDC2 Branku prosljeđuje ključ K_{AB} kako bi on mogao kontaktirati Anu. Dobivena poruka se rastavlja na dijelove te uzimamo određene dijelove iz poruke kako bi ih se moglo proslijediti porukom koja se sljedeća sastavlja. Funkcija „kdc1_kdc2“ je tip string pokazivača jer vraća polje u kojem su obje verzije poruke (dekriptirana i enkriptirana). Ta funkcija označava komunikaciju između KDC1 i KDC2 (KDC_i i KDC_j). Funkciji „kdc2_bob“ se prosljeđuju obje verzije poruke kako bi ih mogli ispisati i poslati enkriptiranu verziju.

```
string* kdc1_kdc2(int clientSocket){
    // Primanje poruke P3 po dijagramu toka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string p3(buffer);
    cout << "Pristigla poruka p3 od KDC1" << endl << "Sadrzaj enkriptirane
poruke: " << p3 << endl << "Dekripcija..." << endl;
    // Dekriptiranje
    string m3Decrypted = caesar(p3, listaKdc[1].key, 0);
    cout << "Sadrzaj citljive poruke: " << m3Decrypted << endl;
```

```
    vector<string> parts = splitString(m3Decrypted, '/');
    string p4 = parts[1] + "/" + parts[0];
    vector<string> parts2 = splitString(parts[2], '.');
    string m4Encrypted = caesar(p4, clientKeys[stoi(parts2[1])], 1);
    static string m4Full[2] = {p4, m4Encrypted};
    // Vrati p4 u kriptiranom i citljivom formatu
    return m4Full;
}

void kdc2_bob(int clientSocket, string p4[]){
    cout << "Saljem poruku p4 za Branka" << endl << "Sadrzaj citljive
poruke: " << p4[0] << endl << "Enkripcija..." << endl;
    cout << "Sadrzaj enkriptirane poruke: " << p4[1] << endl;
    send(clientSocket, p4[1].c_str(), p4[1].length(), 0);
}
```

6.3.4. Programsko rješenje Branko

Brankovo programsko rješenje ćemo zadnje opisati jer on zadnji uspostavlja konekciju s Anom kako bi je autenticirao te kada to izvrši, veza je uspješno uspostavljena. Funkcija „kdc2_bob“ označava komunikaciju između KDC2 i Branka kojim Branko dobije ključ K_{AB} kojim može naknadno kontaktirati Anu i obaviti autentikaciju.

```
void kdc2_bob(int clientSocket){
    // Primanje poruke P4 po dijagramu toka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string p4(buffer);
    cout << "Pristigla poruka p4 od KDC2" << endl << "Sadrzaj enkriptirane
poruke: " << p4 << endl << "Dekripcija..." << endl;
    // Dekripcija
    string m4Decrypted = caesar(p4, key, 0);
    cout << "Sadrzaj citljive poruke: " << m4Decrypted << endl;

    vector<string> parts = splitString(m4Decrypted, '/');
    // Zapisivanje Kab i id Ane u globalne varijable
    kab = stoi(parts[1]);
```

```
    idCommunication = parts[0];
}

void bob_alice(int clientSocket){
    // Branko generira broj (Nonce) koji salje Ani
    int num = rand() % 10000;
    string p5 = to_string(num);
    cout << "Saljem poruku p5 za Anu" << endl << "Sadrzaj citljive poruke:
" << p5 << endl << "Enkripcija..." << endl;
    // Enkripcija
    string m5Encrypted = caesar(p5, kab, 1);
    cout << "Sadrzaj enkriptirane poruke: " << m5Encrypted << endl << endl;
    // Slanje poruke P5 po dijagramu toka
    send(clientSocket, m5Encrypted.c_str(), m5Encrypted.length(), 0);

    // Primanje poruke P6 po dijagramu toka
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string p6(buffer);
    cout << "Pristigla poruka p6 od Ane" << endl << "Sadrzaj enkriptirane
poruke: " << p6 << endl << "Dekripcija..." << endl;
    // Dekripcija
    string m6Decrypted = caesar(p6, kab, 0);
    cout << "Sadrzaj citljive poruke: " << m6Decrypted << endl;
}
```

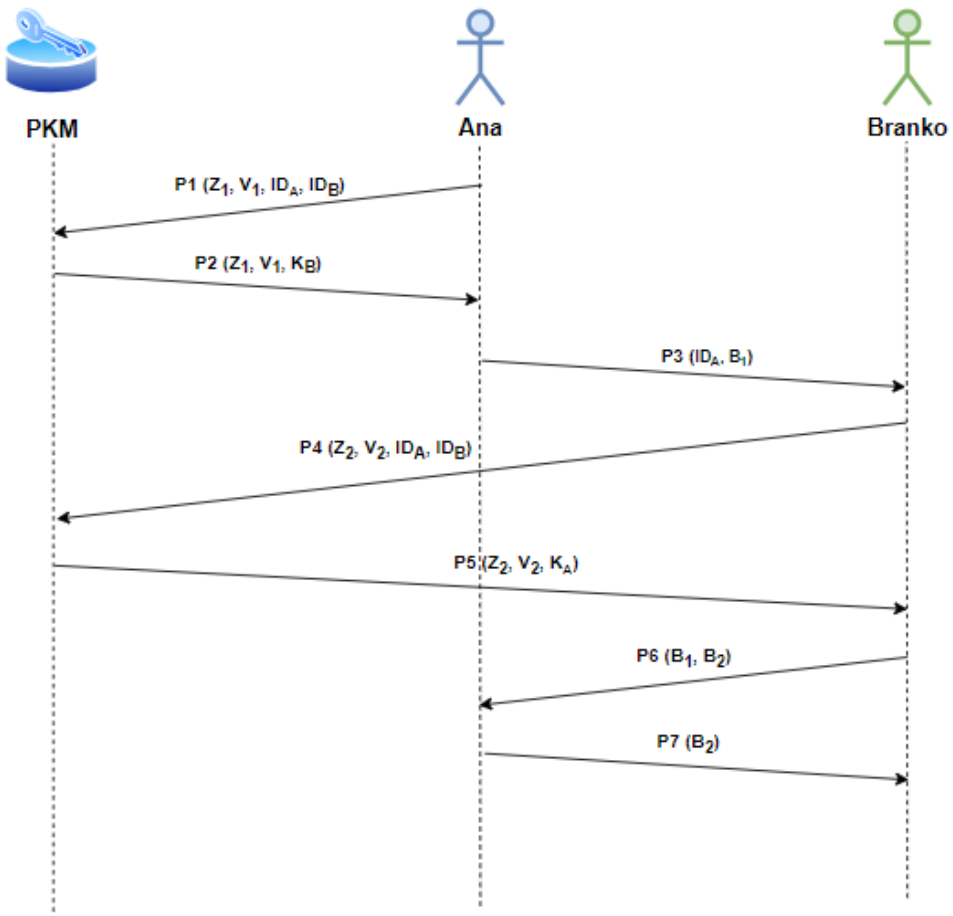
6.4. Implementacija raspodjele ključeva u zatvorenom asimetričnom kriptosustavu

Do sad su u ovom radu obrađene 3 implementacije koje su bazirane na ostvarenju simetričnog kriptosustava. Sljedećom će se implementacijom pokazati kako je moguće razmijeniti sigurnosne ključeve da bi se ostvario asimetričan kriptosustav.

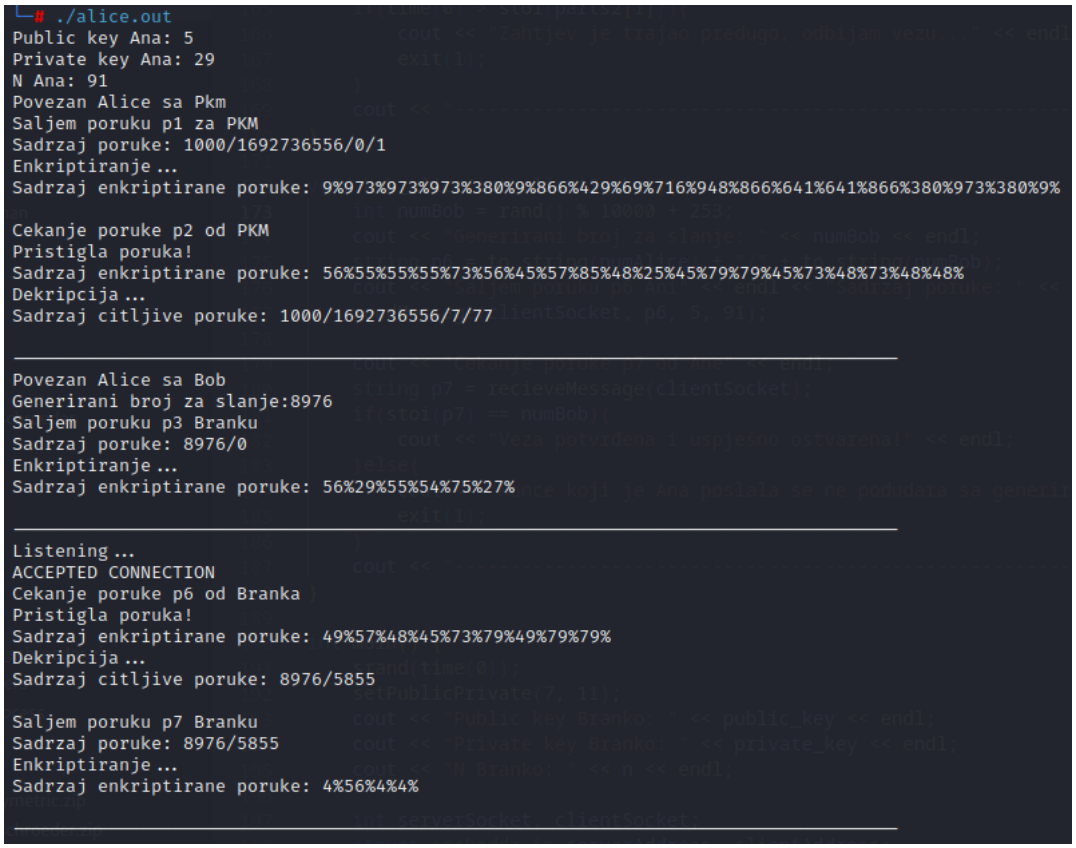
Korisnici Ana i Branko do već su poznati do sad no uvodi se i nova točka komunikacije zvana centar za raspodjelu javnih ključeva (engl. public key manager). U njega su pohranjeni svi javni ključevi korisnika. Kako bi se jedan korisnik povezao s drugim, morat će prvo zatražiti

centar za raspodjelu ključeva kako bi mogao poruku kriptirati javnim ključem tog korisnika i poslati.

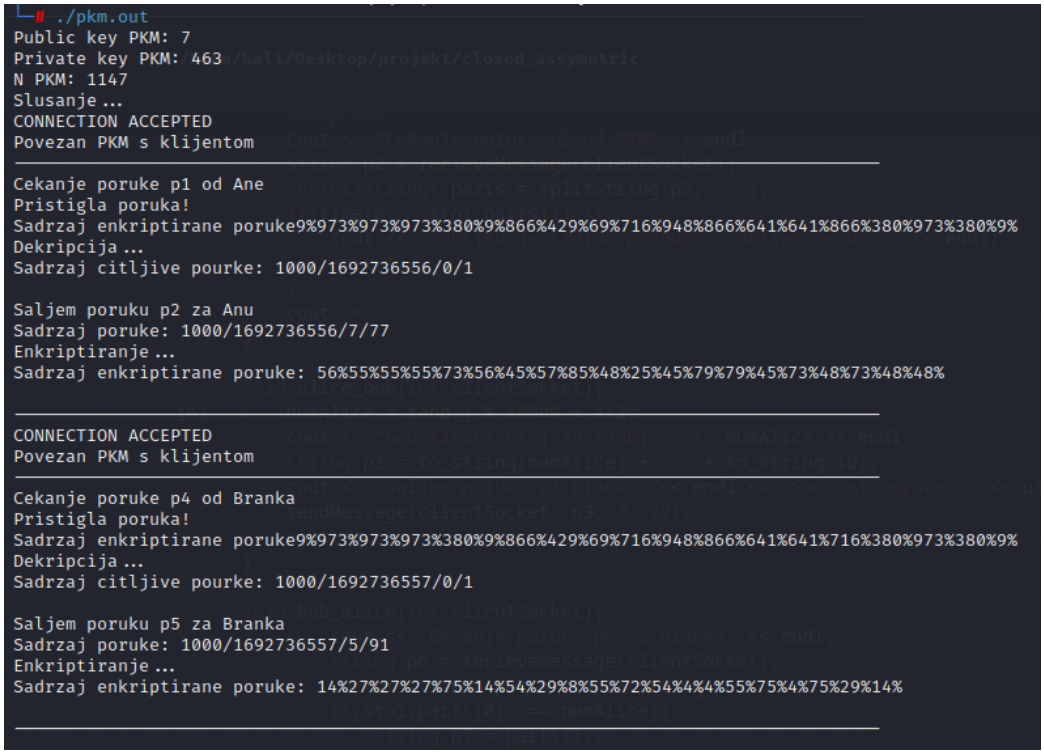
Na slici ispod je prikazan dijagram tijeka poruka ovo protokola. Vidjeti se može kako Ana porukom P_1 započinje prvi korak. U toj poruci PKM-u šalje kod zahtjeva Z_1 kojim on zna što raditi s porukom, vremensku oznaku T_1 te svoju i Brankovu identifikacijsku oznaku (ID_A, ID_B). PKM tada Ani šalje javni ključ Branka K_B, T_1 te Z_1. Ana sada može kontaktirati Branka s njegovim javnim ključem. Šalje mu poruku koja se sastoji od njene identifikacijske oznake ID_A te slučajno generiranog broja B_1. Branko će sada na isti način kontaktirati PKM kako bi mogao dobiti njen javni ključ. To radi porukama P_4 i P_5 te nakon tog procesa može kontaktirati Anu. Porukom P_6 šalje novo generirani broj B_2 te broj koji je Ana ranije generirala (B_1). Ana tada može autenticirati da je to uistinu Branko kada usporedi broj B_1 koji je ranije poslala s brojem B_2 koji je primila. Ukoliko su brojevi jednaki tada broj B_2 natrag šalje Branku kako bi i on mogao potvrditi vezu.



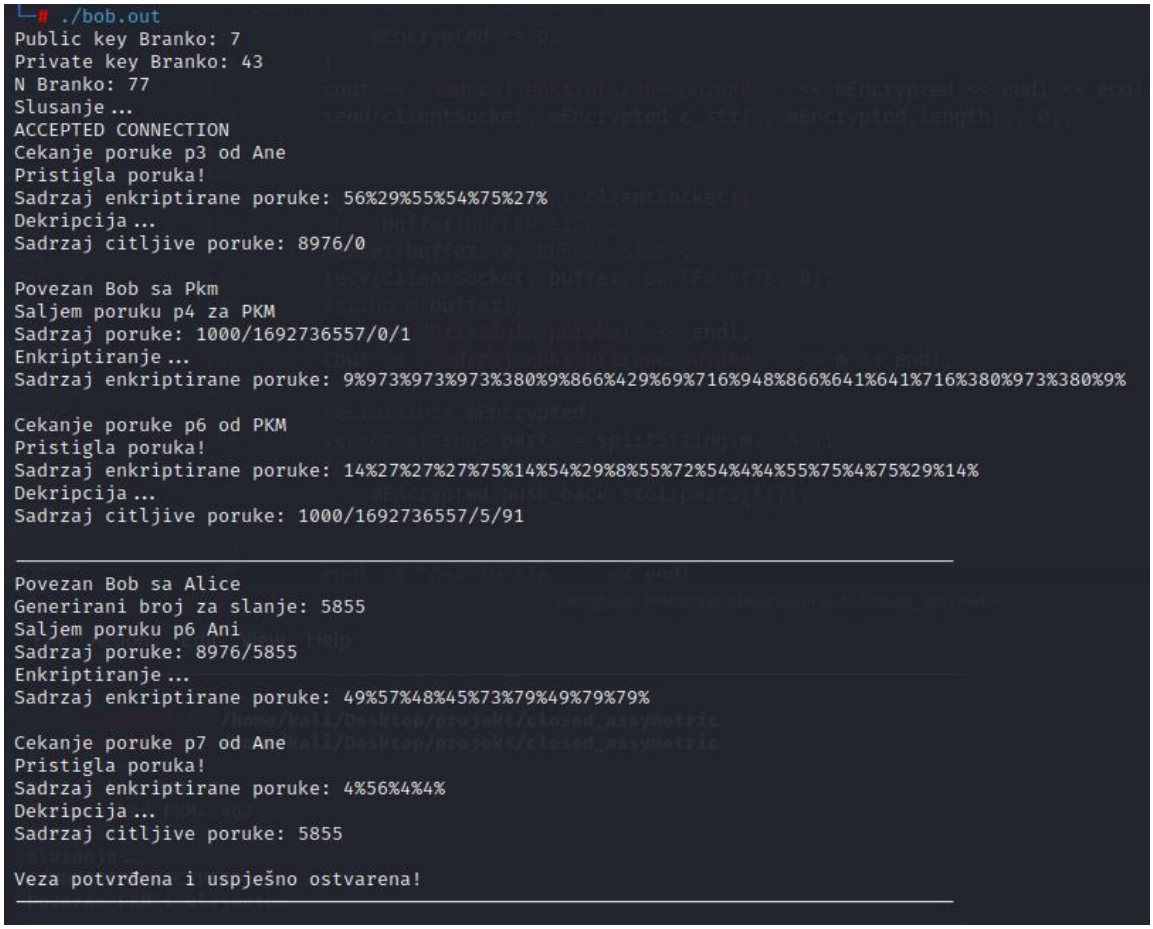
Slika 33: Tijek poruka protokola (Prema: Budin, Golub, Jakobović i Jelenković, 2016)



Slika 34: Rezultat uspostave veze (Ana)



Slika 35: Rezultat uspostave veze (PKM)



Slika 36: Rezultat uspostave veze (Branko)

6.4.1. RSA algoritam

Implementacija asimetričnog kriptosustava zahtijeva da se za enkripciju podataka koristi algoritam asimetrične enkripcije podataka. Iz tog razloga u ovoj implementaciji koristi se pojednostavljeni prikaz RSA enkripcije i dekripcije podataka. Sljedeći isječak programskog koda prikazat će programsko rješenje RSA algoritma.

Možemo vidjeti da se privatni i javni ključ postavljaju funkcijom „setPublicPrivate“ koja za argumente prima brojeve p i q kako bi mogli izračunati ključeve.

```
void setPublicPrivate(int p, int q)
{
    n = p * q;
    int fi = (p - 1) * (q - 1);
```

```
    int e = 2;
    while (1) {
        if (__gcd(e, fi) == 1)
            break;
        e++;
    }
    pbKey = e;
    int d = 2;
    while (1) {
        if ((d * e) % fi == 1)
            break;
        d++;
    }
    prKey = d;
}
```

Funkcija „encrypt“ predstavlja dio RSA algoritma koji je predviđen za enkripciju podataka koje ćemo primiti porukama. Nakon što funkcija završi s enkripcijom vraća rezultat natrag kako bismo ga dalje mogli koristiti.

```
vector<string> encrypt(string message, int key, int nn)
{
    vector<string> form;
    long long int encLetter = 1;
    for (auto& letter : message){
        int e = key;
        encLetter = 1;
        while (e--) {
            encLetter *= (int)letter;
            encLetter %= nn;
        }

        form.push_back(to_string(encLetter));
        form.push_back("%");
    }
    return form;
}
```

Isto tako postoji i funkcija za dekriptiranje podataka koja na sličan način radi dekodiranje podataka kako bismo ih mogli pročitati i dalje obraditi. Svako slovo se enkriptira i dekriptira zasebno a to možemo i vidjeti for funkcijom koja za svaki broj u vektoru „encryptedMessage“ izvodi algoritam dekripcije.

```
string decrypt(vector<int> encryptedMessage)
{
    string s;
    for (auto& num : encryptedMessage){
        int d = prKey;
        long long int decLetter = 1;
        while (d--) {
            decLetter *= num;
            decLetter %= n;
        }
        s += decLetter;
    }
    return s;
}
```

6.4.2. Slanje poruka

Funkcija „sendMessage“ je funkcija kojom ispisujemo da enkriptiramo poruku i šaljemo je sudioniku komunikacije.

```
void sendMessage(int clientSocket, string poruka, int key, int part){
    cout << "Enkriptiranje..." << endl;
    vector<string> coded = encrypt(poruka, key, part);
    string mEncrypted = "";
    for (auto& p : coded){
        mEncrypted += p;
    }
    cout << "Sadrzaj enkriptirane poruke: " << mEncrypted << endl << endl;
    send(clientSocket, mEncrypted.c_str(), mEncrypted.length(), 0);
}
```

6.4.3. Primanje poruka

Kao i „sendMessage“, funkcija „recieveMessage“ služi nam za obradu poruke odnosno njome obavještavamo da je poruka pristigla te ispisujemo dekriptiranu.

```
string recieveMessage(int clientSocket){
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    recv(clientSocket, buffer, BUFFER_SIZE, 0);
    string m(buffer);
    cout << "Pristigla poruka!" << endl;
    cout << "Sadrzaj enkriptirane poruke: " << m << endl;

    vector<int> mEncrypted;
    vector<string> parts = splitString(m, '%');
    for(int i=0; i<parts.size(); i++){
        mEncrypted.push_back(stoi(parts[i]));
    }

    cout << "Dekripcija..." << endl;
    string mDecrypted = decrypt(mEncrypted);
    cout << "Sadrzaj citljive poruke: " << mDecrypted << endl << endl;

    return mDecrypted;
}
```

6.4.4. Korištenje dretva

U ovoj implementaciji na server se povezuju više klijenata i za svakog server izvodi neku funkciju pa je iz tog razloga korištenje dretava neizbježno. Dretve u ovom slučaju pomažu kako bi server svakog klijenta mogao zasebno i nesmetano obrađivati. Petlja se trenutno izvodi 2 puta jer je namještena samo za ovaj primjer ali to se lako može promijeniti ukoliko ćemo u budućnosti zahtijevati više.

```
   std::vector<std::thread> threads;


    int c = 0;
    while (c < 2) {
        // Prihvacanje nadolazece veze i kreiranje socketa te ga se smjesta
u clientSocket
        clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddr,
&clientAddrLen);
        if (clientSocket == -1) {
            cerr << "Pogreška kod prihvačanja konekcije!" << endl;
            continue;
        }else{
            cout << "CONNECTION ACCEPTED" << endl;
        }


        // Ovisno o koraku ispisujemo s kojim se klijentom povezujemo
            cout << "Povezan PKM s klijentom" << endl;
            cout << "-------------------------------------------------------
------------------------------" << endl;
            threads.emplace_back(accept_client, clientSocket, c);
            sleep(5);
            cout << "-------------------------------------------------------
------------------------------" << endl;


        // Odvoji dretvu
        threads.back().detach();
        c++;
    }


    // Zatvori socket
    close(serverSocket);
```

7. Zaključak

Radom je teoretski obrađeno poglavlje kriptografije odnosno ostvarenje kriptirane konekcije putem implementacije sigurnosnih protokola za razmjenu tajnog ključa. Prilikom izrade rada, korišteni su razni izvori u raznim oblicima kako bi se moglo sa što većom sigurnošću istaknuti najbitnije točke kod razmjene ključeva. Operacijski sustav Linux omogućio je korištenje raznih biblioteka za mrežno programiranje u programskom jeziku C++ kojima se ostvaruje slanje paketa i komuniciranje programa preko interneta. Svako programsko rješenje uređivano je preko univerzalnog Microsoft Visual Studio Code uređivača teksta.

Obrađeni su algoritmi koji se često koriste kod ostvarenja nekih sigurnosnih protokola te je na mrežnim protokolima raznih OSI slojeva prikazano kako su implementirani takvi sigurnosni protokoli. Praktični dio rada posebice je posvećen tomu kako se na razne načine mogu implementirati sigurnosni protokoli kojima će se razmijeniti tajni ključ ili ključevi i tako ostvariti simetrični ili asimetrični kriptosustav. Taj se kriptosustav koristi za enkriptiranje poruka koje se tada nalaze u sigurnosnom komunikacijskom kanalu i zaštićene su od neprijatelja.

Nakon čitanja ovog rada, može se zaključiti da se komunikacija internetom odvija stalno i to nije jednostavan proces koji se izvodi u jednom koraku. Ostvarenje veze s nekim klijentom ili serverom nije jednostavan koncept već se unutar samog povezivanja nalaze razni okviri, protokoli i algoritmi koji se izvršavaju tijekom povezivanja. Sigurnost na internetu nikad neće biti savršena no u današnjem svijetu mnogi algoritmi i protokoli već su dovoljno dobro matematički razrađeni da možemo u većini slučajeva biti sigurni od neprijatelja. Informacijskoj sigurnosti studente i učenike trebalo bi podučavati u školama sve više i više jer se svijet sve većom brzinom kreće prema kompletnoj digitalizaciji. Što je više digitalnih uređaja i sustava, to nastaje i više problema kojih se mora riješiti kako bi svi bili zaštićeni od krađe informacija.

Popis literature

[1]. L. Budin, M. Golub, D. Jakobović i L. Jelenković, *Operacijski sustavi*, 4. izd., str. 322-334, Zagreb: Element, 2016.

[2]. „Symmetric-key algorithm“, (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Symmetric-key_algorithm [pristupano 06.07.2022.]

[3]. „Public-key cryptography“, (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Public-key_cryptography [pristupano 06.07.2022.]

[4]. A. S. Gillis, „Diffie-Hellman key exchange“, 2022. [Na internetu]. Dostupno: https://www.techtarget.com/searchsecurity/definition/Diffie-Hellman-key-exchange [pristupano 07.07.2022.]

[5]. D. Bombal i E.Harmoush, (25.03.2022.) „TLS Handshake Deep Dive and decryption with Wireshark“ , Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=25_ftpJ-2ME&ab_channel=DavidBombal [pristupano 08.07.2022.]

[6]. „Cihper Suites in TLS/SSL“ (14.07.2023.). Microsoft [Na internetu]. Dostupno: https://learn.microsoft.com/en-us/windows/win32/secauthn/cipher-suites-in-schannel [pristupano 08.07.2022.]

[7]. Cloudshark (bez dat.) Captures [Na internetu]. Dostupno: https://www.cloudshark.org/captures/ff740838f1c2 [pristupano 10.07.2022]

[8]. R. Molenaar, „IPsec“, bez dat. [Na internetu]. Dostupno: https://networklessons.com/cisco/ccie-routing-switching/ipsec-internet-protocol-security [pristupano 10.07.2022.]

[9]. „Internet Key Exchange“ (16.06.2023). Juniper [Na internetu]. Dostupno: https://www.juniper.net/documentation/us/en/software/junos/vpn-ipsec/topics/topic-map/security-ike-basics.html [pristupano 11.07.2022.]

[10]. Sikandar Shaik, (01.01.2021.) „003 IKEv2 Phase1 IKE SA INIT AUTH“, Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=2z36IedvdZ8&list=PLJqb_j53o7Bi1TAId5vNomraeig-9_XuJ&index=3&ab_channel=SikandarShaik [pristupano 11.07.2022.]

[11]. R. Jones, „SSH Handshake Explained“, 2022. [Na internetu] Dostupno: https://goteleport.com/blog/ssh-handshake-explained/ [pristupano 13.07.2022.]

[12]. CBTVid, (20.01.2021.) „Cryptography Basics – SSH Protocol Explained“, Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=0Sffl7YO0aY&ab_channel=CBTVid [pristupano 14.07.2022.]

Popis slika