UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET

IMPLEMENTACIJA BEZBEDNOG SISTEMA ZA RAZMENU PORUKA ZASNOVANOG NA RABBITMQ-U SA PRIMENOM TLS-A I RSA KRIPTOGRAFIJE

- SEMINARSKI RAD -

Studijski program: Računarstvo i informatika
Modul: Bezbednost računarskih sistema
Predmet: Projektovanje i implementacija sigurnog softvera

Student:

Andrea Valčić, br. Ind. 1928

Profesori:

prof. dr Dragan Janković

prof. dr Petar Rajković

Niš, Januar 2026. godine

SADRŽAJ

1. UVOD 3

2. KONCEPT RAČUNARSKE BEZBEDNOSTI I KRIPTOGRAFIJE 3

2.1. Šta je računarska bezbednost? 3
2.2. Izazovi računarske bezbednosti 4
2.3. Uloga kriptografije u računarskoj bezbednosti 5

3. RAZVOJNO OKRUŽENJE, PROGRAMSKI JEZIK I TEHNOLOGIJE KORIŠĆENE ZA IZRADU PRAKTIČNOG DELA 6

3.1. Visual Studio Code 6
3.2. Python 6
3.3. TLS 7
3.3.1 TLS u odnosu na SSL i HTTPS *7*
3.3.2 TLS sertifikat *8*
3.3.3 Kako funkcioniše TLS? *8*
3.4. RSA algoritam 9
3.4.1 Kako funkcioniše RSA algoritam ? *9*
3.4.2 Ko koristi RSA enkripciju ? *9*
3.4.3 Kritične ranjivosti RSA algoritma *10*
3.5. Sign-then-Encrypt šema 10
3.6. RabbitMQ 11
3.6.1 Šta je RabbitMQ? *11*
3.6.2 AMQP *11*
3.6.3 RabbitMQ komponente *11*
3.6.4 Prednosti korišćenja RabbitMQ-a *12*

4. IMPLEMENTACIJA PRAKTIČNOG DELA 13

4.1. Arhitektura sistema 13
4.2. Implementacija RabbitMQ servera sa TLS-om 13
4.2.1 Kreiranje sertifikata i podešavanje TLS-a *13*
4.2.2 Kreiranje RabbitMQ korisnika *14*
4.2.3 Povezivanje na RabbitMQ preko TLS-a *14*
4.3. Implementacija servera 15
4.4. Implemetacija klijentske aplikacije 16

5. ZAKLJUČAK 18

LITERATURA 19

1. UVOD

Razvoj sigurnog softvera predstavlja jedan od ključnih zadataka savremenog softverskog inženjerstva, posebno u sistemima koji podrazumevaju razmenu podataka između više korisnika. U distribuiranim sistemima često se javljaju problemi pouzdane autentifikacije učesnika i poverljivosti poruka, što može dovesti do neovlašćenog pristupa i zloupotrebe podataka. Motivacija za ovaj rad proizilazi iz potrebe da se prilikom projektovanja i implementacije softvera bezbednost integriše kao osnovni zahtev, a ne kao naknadno dodata funkcionalnost. Cilj seminarskog rada je da predstavi korake u projektovanju i implementaciji sigurnog sistema za razmenu poruka, koji koristi kriptografske mehanizme za zaštitu komunikacije. Praktični deo rada obuhvata realizaciju sistema zasnovanog na RabbitMQ posredniku, TLS protokolu i RSA kriptografiji, sa primenom Sign-then-Encrypt šeme radi obezbeđivanja poverljivosti, integriteta i autentičnosti poruka.

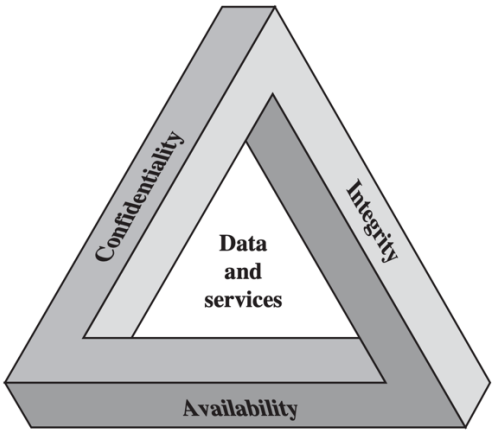
2. Koncept računarske bezbednosti i kriptografije

Bezbednost softvera je važan deo razvoja savremenih informacionih sistema, jer je softver srž funkcionisanja i upravljanja podacima u sistemu. Pretnje upućene softveru danas često iskorišćavaju slabosti u njegovom dizajnu ili implementaciji, zbog čega bezbednost mora biti sastavni deo procesa razvoja, a ne da se tretira kao dodatak. Projektovanje sigurnog softvera znači predvideti moguće napade i ugraditi mehanizme koji čuvaju poverljivost, integritet i dostupnost podataka, čak i kada sistemi rade u kompleksnim i distribuiranim okruženjima.

2.1. Šta je računarska bezbednost?

Računarska bezbednost predstavlja zaštitu informacionih sistema i njihovih resursa, uključujući hardver, softver, podatke i komunikacione tehnologije, radi ostvarivanja osnovnih bezbednosnih ciljeva sistema. Osnovni ciljevi računarske bezbednosti poznati su kao CIA triada:

- Poverljivost (Confidentiality): obezbeđuje da podaci i informacije budu dostupni samo ovlašćenim korisnicima i štiti privatnost korisnika i poverljive informacije organizacije
- Integritet (Integrity): garantuje da se informacije i programi menjaju isključivo na predviđen i ovlašćen način i da sistem funkcioniše bez neovlašćenih ili nenamernih modifikacija
- Dostupnost (Availability): osigurava pravovremen i pouzdan pristup informacijama i servisima ovlašćenim korisnicima, bez prekida ili ograničenja



Slika 1. CIA triada

Pored osnovne CIA triade, savremeni sistemi računarske bezbednosti često uključuju i dodatne principe koji proširuju i jačaju zaštitu sistema:

- Autentičnost: omogućava proveru identiteta korisnika i pouzdanost poruka ili transakcija, tj. potvrđuje da su korisnici i podaci ono što tvrde da jesu.
- Odgovornost: obezbeđuje praćenje aktivnosti korisnika i pripisivanje odgovornosti za eventualne bezbednosne incidente. Ovaj princip podržava neporicanje, detekciju i prevenciju upada, izolaciju grešaka, forenzičku analizu i eventualne pravne postupke.

2.2. Izazovi računarske bezbednosti

Računarska bezbednost predstavlja složenu oblast koja obuhvata veliki broj tehničkih i organizacionih izazova. Poverljivost, integritet i dostupnost su osnovni bezbednosni ciljevi koji na prvi pogled deluju jednostavno, ali mehanizmi za njihovu realizaciju su često složeni i zahtevaju dobro razumevanje mogućih pretnji i napada.

Jedan od glavnih izazova u oblasti računarske bezbednosti jeste činjenica da se bezbednosni mehanizmi moraju projektovati tako da na umu uvek treba imati scenarije potencijalnih napada. Napadači često koriste neočekivane pristupe i pronalaze slabosti koje nisu bile očigledne tokom dizajna sistema. Zbog toga su bezbednosna rešenja često znatno složenija nego što bi se moglo zaključiti na osnovu samih zahteva.

Dodatni izazovi u računarskoj bezbednosti uključuju sledeće:

- Složenost mehanizama zaštite: Bezbednosni algoritmi i protokoli često zahtevaju napredne matematičke i logičke koncepte, što otežava njihovu pravilnu implementaciju i razumevanje.

- Određivanje mesta primene bezbednosti: Neophodno je odlučiti gde i na kom nivou sistema primeniti određene bezbednosne mehanizme, bilo da se radi o fizičkoj lokaciji u mreži ili o slojevima softverske arhitekture.

- Upravljanje tajnim podacima: Bezbednosni sistemi se često oslanjaju na tajne informacije, kao što su kriptografski ključevi, što povlači za soboomo i pitanja o njihovin generacijama, distribuciji, skladištenju i zaštiti.

- Zavisnost od komunikacionih sistema: Kašnjenja i nepredvidivo ponašanje mrežnih protokola mogu otežati primenu određenih bezbednosnih mehanizama, posebno onih koji zavise od vremenskih ograničenja.

- Asimetrija između napadača i branioca: Napadaču je često dovoljno da pronađe jednu slabost u sistemu, dok osoba zadužena za dizajniranje sistema mora identifikovati i otkloniti sve potencijalne propuste kako bi sistem bio bezbedan.

- Nedovoljna svest o značaju bezbednosti: Korisnici i administratori često ne vide direktnu korist od ulaganja u bezbednost dok ne dođe do bezbednosnog incidenta.

- Potreba za stalnim nadzorom: Efikasna bezbednost zahteva kontinuirano praćenje sistema, što je teško ostvarivo u okruženjima sa ograničenim resursima.

- Bezbednost kao naknadna aktivnost: Bezbednost se često dodaje sistemu nakon završetka razvoja, umesto da bude integralni deo procesa projektovanja.

- Uticaj na upotrebljivost: Jaki bezbednosni mehanizmi se ponekad doživljavaju kao prepreka jednostavnom i efikasnom korišćenju sistema.

2.3. Uloga kriptografije u računarskoj bezbednosti

Kriptografija ima ključnu ulogu u računarskoj bezbednosti jer omogućava zaštitu podataka u softverskim sistemima koji rade u okruženjima u kojima nije moguće u potpunosti verovati svim učesnicima i komunikacionim kanalima. Njena osnovna svrha je da obezbedi sigurnu razmenu i čuvanje informacija tako što će sprečiti neovlašćen pristup i neovlašćene izmene podataka.

Jedan od osnovnih kriptografskih mehanizama jeste šifrovanje, čija je uloga da obezbedi poverljivost podataka. Originalni sadržaj poruke koji prođe kroz proces šifrovanja se transformiše u oblik koji neovlašćene strane ne mogu razumeti jer je nečitljiv i time se sprečava otkrivanje osetljivih informacija tokom skladištenja ili prenosa. Na ovaj način, čak i ako dođe do presretanja podataka, njihov sadržaj ostaje zaštićen.

Sa druge strane, digitalni potpis ima podjednako važnu, ali drugačiju ulogu. Njime se obezbeđuje autentičnost i integritet poruke, kao i mogućnost da se potvrdi identitet njenog pošiljaoca. Digitalni potpis omogućava proveru da li je poruka izmenjena tokom prenosa i pruža dokaz o poreklu poruke, čime se smanjuje mogućnost lažnog predstavljanja i negiranja izvršenih radnji.

Kriptografija predstavlja osnov sigurnog softvera jer omogućava ostvarivanje ključnih bezbednosnih ciljeva, kao što su poverljivost, integritet, autentičnost i neporicanje. Bez njenih mehanizama, softverski sistemi ne bi mogli da pruže pouzdanu zaštitu podataka niti da garantuju bezbednu komunikaciju između korisnika i komponenti sistema. Zbog toga predstavlja neizostavan deo projektovanja i implementacije sigurnog softvera i temelj na kome se zasnivaju savremena bezbednosna rešenja.

3. Razvojno okruženje, programski jezik i tehnologije korišćene za izradu praktičnog dela

3.1. Visual Studio Code

Tokom razvoja sistema korišćeno je programsko okruženje Visual Studio Code, koje pruža podršku za veliki broj programskih jezika. Ovo okruženje se široko primenjuje kako u razvoju veb aplikacija, tako i u izradi aplikacija za mobilne uređaje i tablete. Visual Studio Code dolazi sa ugrađenom podrškom za JavaScript, TypeScript i Node.js, dok se dodatna funkcionalnost obezbeđuje kroz bogat ekosistem ekstenzija za druge programske jezike i izvršna okruženja, kao što su C++, C#, Java, Python, PHP, Go i .NET. Pored toga, okruženje omogućava povezivanje na udaljene mašine putem SSH protokola, što je značajno olakšalo rad sa serverskom komponentom sistema.

3.2. Python

Prilikom izrade sistema za bezbednu razmenu poruka korišćen je Python programski jezik. Ovaj programski jezik je razvio Gvido van Rosum početkom devedeseth godina i predstavlja programski jezik visokog nivoa opšte namene. Podržava imperativni, objektno-orjentisan i funkcionalni stil programiranja. Python se odlikuje preglednom i jednostavnom sintaksom, što ga čini pogodnim za brzo učenje i razvoj aplikacija. Programi napisani u ovom jeziku izvršavaju se pomoću interpretatora, uz koji dolazi i obimna standardna biblioteka modula.

3.2.1 Biblioteke u Python-u

Python raspolaže bogatim ekosistemom modula i paketa koji nude širok spektar funkcionalnosti i omogućavaju programerima da realizuju različite zadatke bez potrebe za pisanjem koda od nule. Ove biblioteke sadrže unapred definisane klase, funkcije i rutine koje se koriste za razvoj aplikacija, automatizaciju procesa, obradu i analizu podataka, matematičke proračune i druge složene operacije. Ekosistem Python biblioteka obuhvata brojne oblasti, uključujući razvoj veb aplikacija, analizu podataka, mašinsko učenje, obradu slika i naučna računanja, što značajno doprinosi popularnosti ovog programskog jezika i olakšava efikasnu implementaciju kompleksnih rešenja.

Python biblioteke korišćene u praktičnom delu rada:

- **FastAPI**: Omogućava kreiranje web servera i API-ja koji prihvataju i šalju poruke između klijenata i servera. Ključna je za definisanje REST ili WebSocket endpoint-a za razmenu poruka.

- **Requests**: Klijentima ili serverima omogućava slanje HTTP zahteva drugim servisima ili mikroservisima.

- **SSL**: Omogućava TLS/SSL zaštitu veze između klijenta i servera, što je neophodno za sigurnu transportnu komunikaciju.

- **Cryptography**: Osnova za end-to-end enkripciju i digitalno potpisivanje poruka. serialization za čuvanje i učitavanje ključeva, hashes i sha256 za verifikaciju integriteta, rsa i padding za sigurno šifrovanje i potpisivanje.

- **Hashlib**: Za kreiranje kriptografskih sažetaka (hash) podataka, što pomaže da se proveri integritet poruka.

- **Pydantic**: Koristi se za validaciju i serijalizaciju podataka koji se šalju i primaju, što je važno da se spreče neispravni ili maliciozni podaci.

- **Pika**: Omogućava povezivanje sa RabbitMQ serverom, koji služi za pouzdanu i asinhronu razmenu poruka između korisnika. Ključno za implementaciju reda poruka i distribucije između klijenata.

3.3. TLS

Transport Layer Security (TLS) je bezbednosni protokol koji obezbeđuje privatnost i zaštitu podataka tokom komunikacije preko Interneta. Njegova osnovna funkcija je enkripcija komunikacije između aplikacija i servera, ali se koristi i za zaštitu e-pošte, razmenu poruka i VoIP servisa. TLS je standardizovan od strane IETF-a, prva verzija je objavljena 1999, a najnovija verzija TLS 1.3 od 2018. godine pruža najefikasniju i najbezbedniju zaštitu prenosa podataka.

TLS protokol obezbeđuje sigurnu komunikaciju kroz tri ključna aspekta: enkripciju, autentifikaciju i integritet podataka.

- Enkripcija štiti informacije tako da ih neovlašćene osobe ne mogu pročitati.

- Autentifikacija potvrđuje identitet učesnika u komunikaciji.

- Integritet garantuje da podaci nisu izmenjeni ili falsifikovani tokom prenosa.

3.3.1 TLS u odnosu na SSL i HTTPS

TLS predstavlja unapređenje starijeg protokola SSL (Secure Sockets Layer), koji je razvila kompanija Netscape. Verzija TLS 1.0 je u početku razvijana kao SSL 3.1, ali je naziv promenjen pre objavljivanja kako bi se naglasilo da protokol više nije povezan sa Netscape-om. Zbog ove istorijske povezanosti, termini TLS i SSL se ponekad koriste naizmenično, iako je TLS modernija i sigurnija verzija protokol..

Često se postavlja pitanje koja je razlika između TLS-a I HTTPS-a. Razlika je u tome što je HTTPS implementacija TLS enkripcije na HTTP protokolu, što znači da svi sajtovi koji koriste HTTPS zapravo koriste TLS za zaštitu podataka tokom prenosa, dok običan HTTP ne pruža nikakvu enkripciju niti zaštitu komunikacije.

3.3.2 TLS sertifikat

Da bi web sajt ili aplikacija koristili TLS, na svom serveru moraju imati instaliran TLS sertifikat (poznat i kao SSL sertifikat zbog terminološke zabune). Sertifikat izdaje ovlašćena sertifikaciona institucija i sadrži podatke o vlasniku domena, kao i javni ključ servera, što omogućava verifikaciju identiteta servera i uspostavljanje sigurne komunikacije.

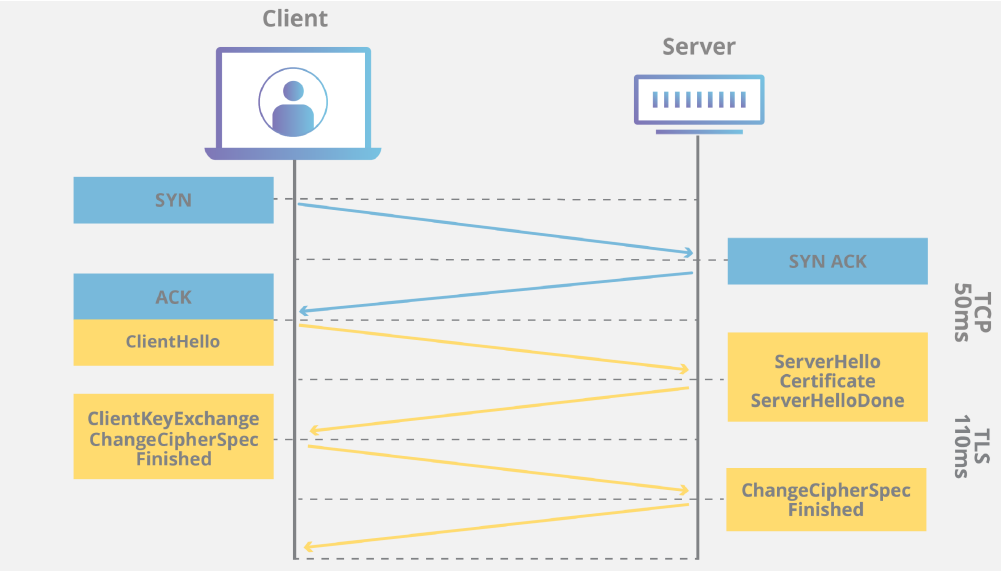
3.3.3 Kako funkcioniše TLS?

TLS funkcioniše kroz proces koji se naziva TLS rukovanje (handshake), a započinje u trenutku kada korisnik pristupi veb sajtu ili aplikaciji koja koristi ovaj protokol. Tokom ovog procesa, korisnički uređaj i server međusobno uspostavljaju siguran komunikacioni kanal.

U okviru TLS rukovanja, strane se dogovaraju o verziji TLS protokola koja će se koristiti, kao i o metodama zaštite podataka. Server zatim dokazuje svoj identitet pomoću TLS sertifikata, čime klijent dobija potvrdu da komunicira sa pouzdanim serverom. Nakon toga se generišu privremeni ključevi koji će se koristiti za šifrovanje podataka tokom trajanja te sesije.

Važnu ulogu u ovom procesu ima kriptografija sa javnim ključem, koja omogućava bezbedno uspostavljanje zajedničkih ključeva čak i preko nezaštićenog kanala. Javni ključ servera, koji se nalazi u TLS sertifikatu, omogućava proveru autentičnosti servera.

Nakon uspostavljanja veze, svi podaci koji se razmenjuju postaju šifrovani i dodatno zaštićeni mehanizmima koji omogućavaju proveru integriteta. Na taj način primalac može biti siguran da podaci nisu izmenjeni tokom prenosa, što obezbeđuje poverljivu, pouzdanu i bezbednu komunikaciju.



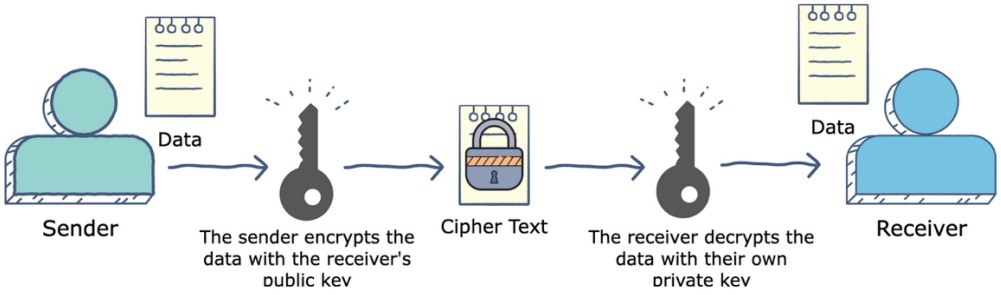
Slika 2. Tok komunikacije između klijenta i servera uz TLS zaštitu

3.4. RSA algoritam

RSA (Rivest-Shamir-Adleman) predstavlja kriptografski algoritam koji je razvijen 1977. godine i koji se i danas široko primenjuje u različitim oblastima računarskih sistema. U nastavku će biti objašnjen njegov princip rada, čime će se pokazati kako se prethodno obrađeni kriptografski algoritmi i koncepti mogu međusobno povezati i koristiti u praktičnim i realnim bezbednosnim rešenjima.

3.4.1 Kako funkcioniše RSA algoritam ?

RSA funkcioniše kao asimetrični kriptografski sistem koji koristi par ključeva: javni i privatni. Kada se podaci šifruju javnim ključem, oni se mogu dešifrovati isključivo odgovarajućim privatnim ključem, što omogućava siguran prenos poverljivih informacija jer samo predviđeni primalac može pristupiti sadržaju poruke. Obrnuti postupak, u kojem se poruka šifruje privatnim ključem, koristi se za potvrdu identiteta pošiljaoca, jer svako ko poseduje njegov javni ključ može proveriti da poruka zaista potiče od njega i da nije izmenjena tokom prenosa.



Slika 3. Ilustracija asimetrične kriptografije

Bezbednost RSA algoritma zasniva se na matematičkom problemu faktorizacije velikih brojeva: iako je lako pomnožiti dva velika prosta broja, izuzetno je teško iz dobijenog proizvoda ponovo odrediti te brojeve. Zbog toga je praktično nemoguće izračunati privatni ključ na osnovu javnog, što RSA čini pouzdanim mehanizmom za zaštitu podataka i autentifikaciju u savremenim informacionim sistemima.

3.4.2 Ko koristi RSA enkripciju ?

RSA enkripciju koriste brojni softverski sistemi i bezbednosni mehanizmi, pre svega za digitalno potpisivanje i verifikaciju identiteta. Jedna od važnih primena je potpisivanje digitalnih sertifikata, gde se privatnim ključem potvrđuje kome pripada određeni javni ključ, čime se obezbeđuje poverenje u komunikaciji.

RSA se takođe koristi za potpisivanje programskog koda, kako bi se proverilo da kod potiče od pouzdanog autora i da nije neovlašćeno izmenjen tokom distribucije. Pored toga, RSA je dugo bio deo TLS protokola za obezbeđivanje komunikacije između klijenta i servera, posebno u fazi uspostavljanja bezbedne veze. Različiti sistemi kao što su VPN servisi, veb pregledači, email servisi i alati za sigurnu razmenu podataka koristili su ili i dalje koriste RSA kao osnovu za autentifikaciju i zaštitu komunikacije.

3.4.3 Kritične ranjivosti RSA algoritma

Iako se RSA i dalje smatra pouzdanim kriptografskim algoritmom, njegova bezbednost u velikoj meri zavisi od pravilne implementacije i izbora odgovarajućih parametara. Nepravilna upotreba može dovesti do različitih bezbednosnih problema, među kojima se izdvajaju sledeće ranjivosti:

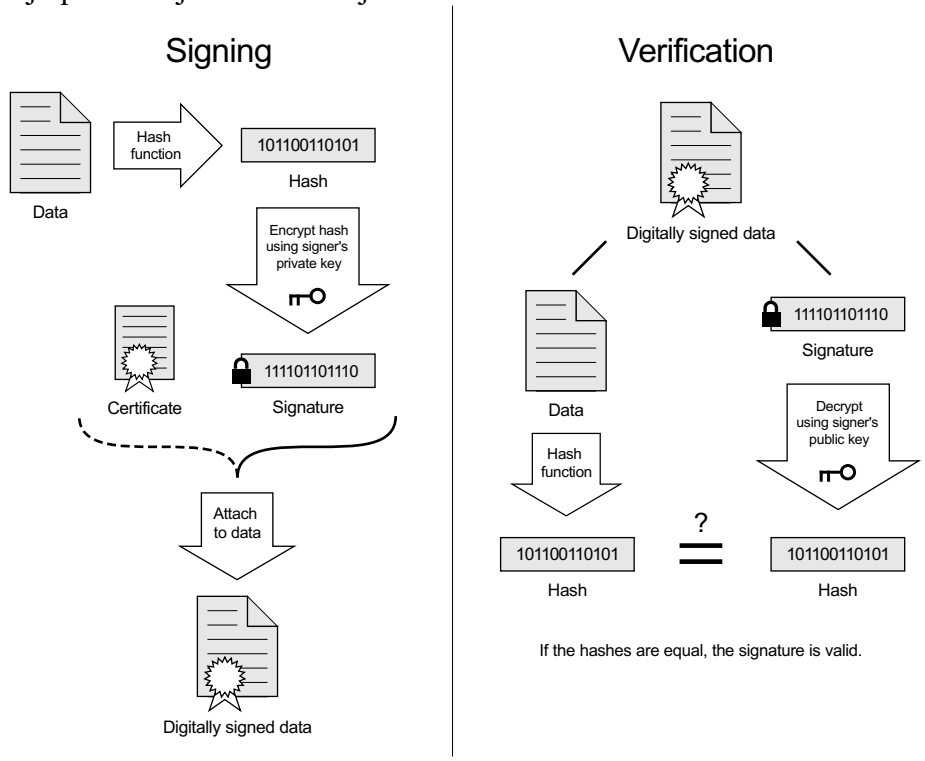
- Slabi generatori slučajnih brojeva
  Ukoliko se za generisanje prostih brojeva koriste nekvalitetni generatori slučajnosti, dobijeni ključevi mogu postati predvidivi i lakši za razbijanje.

- Neispravno generisanje ključeva
  Izbor prostih brojeva koji su previše bliski ili korišćenje neadekvatnih parametara može značajno umanjiti kriptografsku sigurnost RSA algoritma.

- Napadi bočnim kanalima
  Ovi napadi ne ciljaju sam algoritam, već analiziraju ponašanje sistema, kao što su vreme izvršavanja ili potrošnja resursa, kako bi se otkrile poverljive informacije, uključujući privatne ključeve.

3.5. Sign-then-Encrypt šema

Sign-then-Encrypt šema predstavlja kriptografsku tehniku u kojoj se poruka najpre digitalno potpisuje, a zatim se tako potpisana poruka šifruje pre slanja. Na ovaj način se istovremeno obezbeđuju autentičnost i integritet poruke, jer digitalni potpis omogućava primaocu da proveri identitet pošiljaoca i da utvrdi da sadržaj nije menjan, kao i poverljivost, pošto se kompletna poruka zajedno sa potpisom šifruje javnim ključem primaoca. Ovakav redosled operacija je posebno pogodan za distribuirane sisteme i sisteme za razmenu poruka, jer sprečava neovlašćene strane da pristupe sadržaju poruke ili da lažno predstave pošiljaoca, čak i u slučaju presretanja komunikacije.



Slika 4. Dijagram digitalnog potpisa-verifikacije

3.6. RabbitMQ

U asinhronoj komunikaciji, mikroservis komunicira sa drugim mikroservisom tako što asinhrono prosleđuje poruke uz pomoć posrednika, poznatog kao broker poruka. Ovaj pristup se oslanja na obrasce kao što su razmena poruka zasnovana na redu (eng. queue) čekanja i model proizvođača (eng. producer) i potrošača (eng. consumer), koji omogućavaju efikasnu i skalabilnu komunikaciju između mikroservisa.

3.6.1 Šta je RabbitMQ?

RabbitMQ je softver koji funkcioniše kao posrednik u razmeni poruka (eng. message broker). Prvobitno je razvijen za implementaciju AMQP (eng. Advanced Message Queuing Protocol) protokola. RabbitMQ vrši prijem poruka od proizvođača i isporučuje ih potrošačima. Ovaj softver funkcioniše kao srednji sloj (eng. middleware) koji umanjuje opterećenje i vreme isporuke na veb serverima. Takođe, obezbeđuje zajedničku platformu za aplikacije za slanje i primanje poruka tako što ih na bezbedan način čuva u redu dok ne budu uspešno isporučene.

3.6.2 AMQP

AMQP (eng. Advanced Message Queuing Protocol) je protokol za razmenu poruka koji omogućava da klijentske aplikacije koje su u skladu sa AMQP standardom komuniciraju sa brokerima poruka koji su takođe u skladu sa tim protokolom. Brokeri poruka primaju poruke od proizvođača i prosleđuju ih potrošačima. AMQP obuhvata redove, prekidače i veze, poznate kao entiteti, koji obezbeđuju efikasno rutiranje poruka od proizvođača do potrošača.

3.6.3 RabbitMQ komponente

RabbitMQ čine 4 osnovne komponente. Detalji ovih komponenti dati su u nastavku.

Proizvođač (eng. Producer) je program koji šalje poruke. On se povezuje sa RabbitMQ- om, šalje poruku i zatim prekida vezu. Pri slanju poruke, proizvođač može da definiše različite atribute uokviru poruke (metapodatke), koje ponekad obrađuje broker, ali na kraju većinu njih koristi primalac poruke.

Exchange komponenta zajedno sa redom čini broker. Exchange prima poruku od proizvođača i rutira je ka jednom ili više redova. Način rutiranja zavisi od vrste exchange-a i pravila vezivanja. Postoje različite vrste exchange komponenti u RabbitMQ-u:

- Direct exchange: Poruke se šalju u red samo ako se vrednost ključa za rutiranje podudara.
- Fanout exchange: Poruka se šalje svim redovima, bez obzira na ključ rutiranja.
- Topic exchange: Umesto tačnog ključa, koriste se obrasci za rutiranje poruka.
- Header exchange: Poruke se šalju na osnovu vrednosti u zaglavlju koje odgovara ključu za rutiranje.

Red je mesto u RabbitMQ-u gde se poruke skladište. Poruke prolaze kroz RabbitMQ i samu aplikaciju, ali se samo u redu mogu čuvati. Red je ograničen memorijom i prostorom na disku hosta. Više proizvođača može da šalju poruke u isti red, a više potrošača može da prima podatke iz tog reda. Pre korišćenja, red mora biti deklarisan, a ako već postoji, samo se proveravaju njegovi atributi.

Potrošač (eng. Consumer) je program koji čeka da primi poruke iz RabbitMQ-a. Da bi potrošač mogao da prima poruke, red mora biti već kreiran. Kada se potrošač poveže, isporuka poruka počinje odmah ako u redu postoje poruke.

3.6.4 Prednosti korišćenja RabbitMQ-a

Prednosti koje RabbitMQ donosi poslovnim sistemima:

- Asinhrono slanje poruka
  Podržava više protokola za razmenu poruka, redove poruka, potvrde isporuke, fleksibilno rutiranje i razne tipove exchange komponenti.

- Podrška za razvoj
  Mogućnost implementacije uz korišćenje alata kao što su Docker i Puppet, sa podrškom za više programskih jezika, kao što su Java, PHP, Python, JavaScript i drugi.

- Distribuirano postavljanje
  Može da radi na više različitih fizičkih ili virtuelnih lokacija, tzv. zone dostupnosti i regioni. Podržava klasterizaciju, što znači da se više servera može povezati kako bi obezbedili bolje performanse i osigurali da su poruke uvek dostupne.

- Optimizovan za poslovna okruženja i sisteme u oblaku
  Omogućava autorizaciju korisnika, bezbednu komunikaciju preko TLS protokola i integraciju sa LDAP-om za upravljanje korisnicima. Takođe, podržava modularne sisteme za autentifikaciju koji se mogu prilagoditi različitim potrebama. Jednostavan za instaliranje na bilo kojoj platformi ili oblaku.

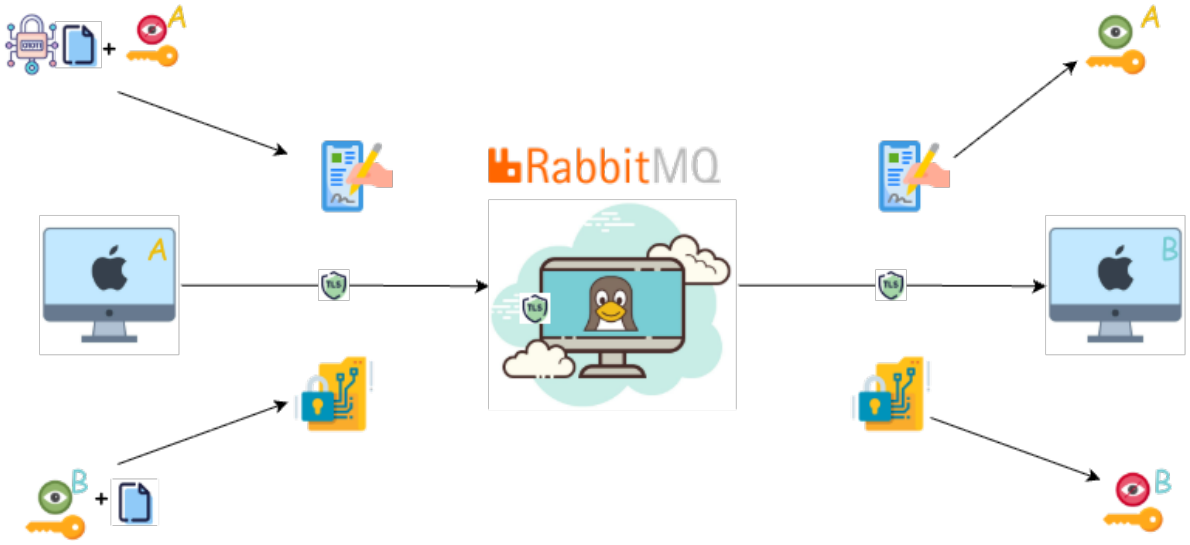
- Alati i dodaci
  Nudi razne alate za kontinuiranu integraciju, operativnu analitiku i integraciju sa drugim poslovnim sistemima. Sistem dodataka predstavlja jedan od najboljih načina za obogaćivanje funkcionalnosti RabbitMQ-a.

- Upravljanje i nadzor
  Interfejs za upravljanje, alati za komandnu liniju i HTTP-API za efektivno upravljanje i nadzor RabbitMQ-a.

4. Implementacija praktičnog dela

4.1. Arhitektura sistema

Arhitektura bezbednog sistema za razmenu poruka predstavlja strukturu i organizaciju komponenti sistema, kao i način na koje ove komponente međusobno komuniciraju. Na slici 5 prikazan je dijagram arhitekture sistema.



Slika 5. Dijagram arhitekture sistema

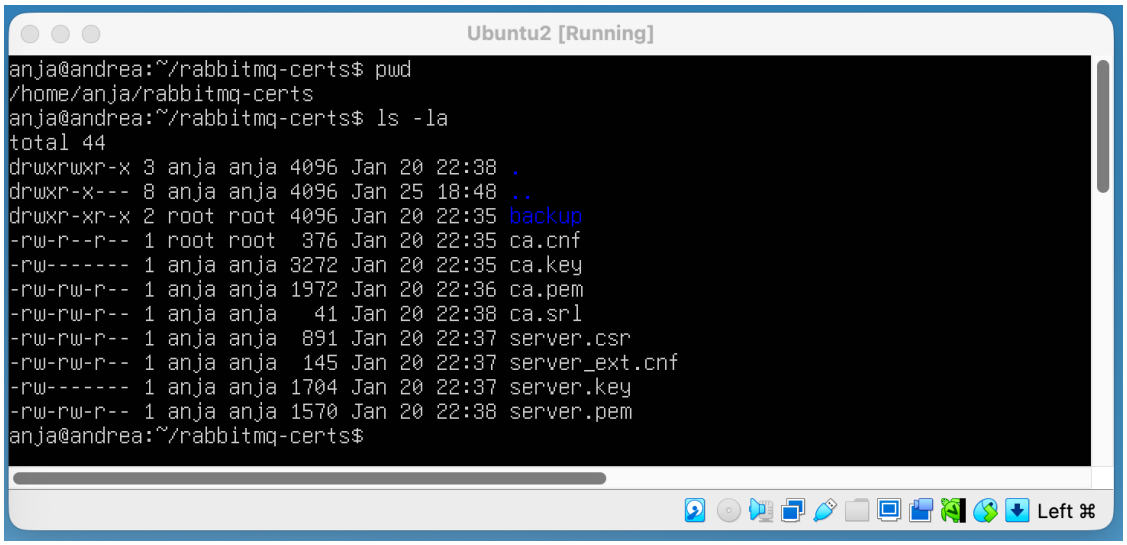
4.2. Implementacija RabbitMQ servera sa TLS-om

Ovo poglavlje opisuje kako je postavljen RabbitMQ sa TLS-om na Linux serveru i kako se macOS klijent bezbedno povezuje i razmenjuje poruke uz dodatnu zaštitu RSA kriptografijom (potpis i šifrovanje).

4.2.1 Kreiranje sertifikata i podešavanje TLS-a

Za TLS komunikaciju kreiran je radni direktorijum **rabbitmq-certs**, u kojem su generisani sledeći fajlovi:

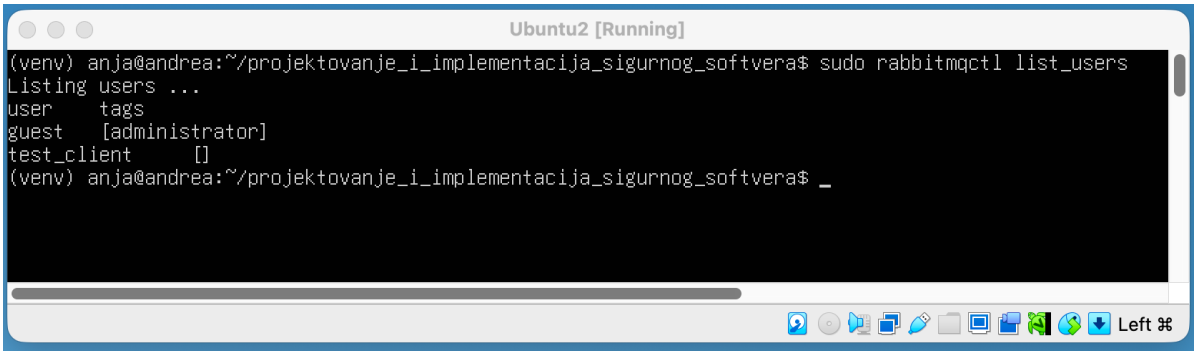
- **CA sertifikat** (ca.pem) - autoriteti sertifikata, koji je kopiran na klijentske uređaje.
- **Serverski privatni ključ** (server.key) - privatni ključ servera.
- **Serverski sertifikat** (server.pem) - sertifikat potpisan od strane interne CA, sa SAN (Subject Alternative Name) koji uključuje ime hosta ili IP adresu. Ovo omogućava klijentima da verifikuju identitet servera prilikom uspostavljanja TLS veze.



Slika 6. Prikaz rabbitmq-certs direktorijuma na serveru

4.2.2 Kreiranje RabbitMQ korisnika

Na serveru je kreiran RabbitMQ korisnik **test_client** koji ima pristup za autentifikaciju klijenata i slanje/primanje poruka.



Slika 7. Prikaz RabbitMQ korisnika na serveru

4.2.3 Povezivanje na RabbitMQ preko TLS-a

Na klijentskoj mašini definisana je metoda čija je namena da uspostavi konekciju na RabbitMQ preko TLS-a. Implementacija funkcije prikazana je na slici u nastavku.

```
def connect_rabbit(username):
    context = ssl._create_unverified_context()
    credentials = pika.PlainCredentials("test_client", "*****")
    params = pika.ConnectionParameters(
        host=RABBITMQ_HOST,
        port=RABBITMQ_PORT,
        ssl_options=pika.SSLOptions(context),
        credentials=credentials
    )
    connection = pika.BlockingConnection(params)
    channel = connection.channel()
    channel.queue_declare(queue=username)
    return connection, channel
```

Slika 8. Metoda za povezivanje na RabbitMQ preko TLS-a

4.3. Implementacija servera

Server u ovom sistemu je implementiran korišćenjem FastAPI framework-a i služi za registraciju korisnika, čuvanje njihovih javnih ključeva i pružanje podataka o registrovanim korisnicima. Funkcionalnosti servera su fokusirane na sigurnu razmenu javnih ključeva između klijenata.

Ključne komponente servera:

1. FastAPI aplikacija
   - Kreirana je instanca FastAPI-a koja omogućava definisanje REST API endpoint-a za komunikaciju sa klijentima.

2. Čuvanje podataka o korisnicima
   - Korišćena je jednostavna Python struktura podataka, rečnik **users,** koji čuva parove **username → javni ključ**.
   - Ova rečnik omogućava serveru da brzo pronađe javni ključ određenog korisnika ili proveri da li korisnik već postoji.

3. REST API endpoint-i
   - **/register (POST)** – Omogućava registraciju novog korisnika. Server prima username i javni ključ putem JSON zahteva i čuva ih u rečniku. Ako korisnik već postoji, vraća se greška.
   - **/key/{username} (GET)** – Vraća javni ključ traženog korisnika. Ako korisnik ne postoji, vraća se HTTP greška 404.
   - **/users (GET)** – Vraća listu svih registrovanih korisnika na serveru.

4. Jednostavnost i fokus na sigurnost ključeva

   Server ne obrađuje sadržaj poruka, već samo upravlja registracijom i distribucijom javnih ključeva. Na ovaj način, server olakšava klijentima da razmenjuju kriptografski zaštićene poruke koristeći sign-then-encrypt šemu.

```
4   app = FastAPI()
5   users = {}
```

Slika 9. Definicija FastAPI aplikacije i rečnika korisnika

```
11  @app.post("/register")
12  def register_user(data: RegisterRequest):
13      if data.username in users:
14          raise HTTPException(status_code=400, detail="User already exists")
15      users[data.username] = data.public_key
16      return f"User '{data.username}' has been registered successfully!"
```

Slika 10. Endpoint za registraciju korisnika:

```python
@app.get("/key/{username}")
def get_public_key(username: str):
    if username not in users:
        raise HTTPException(status_code=404, detail="User not found")
    return {"public_key": users[username]}
```

Slika 11. Endpoint za dobijanje javnog ključa

4.4. Implemetacija klijentske aplikacije

Klijentska aplikacija ima ulogu da omogući korisnicima registraciju, autentifikaciju, sigurnu razmenu poruka i komunikaciju sa RabbitMQ serverom preko TLS-a. Klijent kombinuje kriptografske mehanizme, REST komunikaciju sa serverom i message broker (RabbitMQ) za pouzdanu i bezbednu razmenu poruka.

Ključne funkcionalnosti klijenta:

1. Generisanje i upravljanje kriptografskim ključevima
    - Prilikom registracije, klijent generiše RSA par ključeva (privatni i javni ključ).
    - Privatni ključ se čuva lokalno na klijentu, dok se javni ključ šalje serveru radi kasnije razmene poruka.
    - Ovim pristupom se obezbeđuje da privatni ključ nikada ne napušta klijentsku stranu.
2. Registracija i prijava korisnika
    - Klijent komunicira sa serverom putem REST API-ja kako bi registrovao korisnika i preuzeo informacije o ostalim korisnicima.
    - Lozinka korisnika se lokalno hešira (SHA-256) i čuva u fajlu, dok server ne čuva lozinke, već isključivo javne ključeve.
3. Sigurna komunikacija sa RabbitMQ serverom
    - Klijent se povezuje na RabbitMQ koristeći TLS konekciju, pri čemu se koristi CA sertifikat za verifikaciju servera.
    - Za autentifikaciju se koristi namenski RabbitMQ korisnik (**test_client**).
    - Svaki korisnik ima sopstveni red poruka (**queue**), koji se identifikuje njegovim korisničkim imenom.
4. Slanje poruka (sign-then-encrypt pristup)
    - Poruka se najpre digitalno potpisuje privatnim ključem pošiljaoca, čime se obezbeđuje integritet i autentičnost.
    - Nakon toga se poruka šifruje javnim ključem primaoca, čime se obezbeđuje poverljivost sadržaja.
    - Ovako pripremljena poruka se šalje kroz RabbitMQ red odgovarajućeg korisnika.

5. Prijem i verifikacija poruka

   - Klijent preuzima poruku iz svog reda na RabbitMQ serveru.
   - Poruka se dešifruje privatnim ključem primaoca, a zatim se proverava digitalni potpis koristeći javni ključ pošiljaoca preuzet sa servera.
   - Tek nakon uspešne verifikacije, poruka se prikazuje korisniku.

```python
def generate_keys(username):
    private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
    public_key = private_key.public_key()
    with open(BASE_DIR/f"{username}_private.pem","wb") as f:
        f.write(private_key.private_bytes(
            serialization.Encoding.PEM,
            serialization.PrivateFormat.PKCS8,
            serialization.NoEncryption()
        ))
    with open(BASE_DIR/f"{username}_public.pem","wb") as f:
        f.write(public_key.public_bytes(
            serialization.Encoding.PEM,
            serialization.PublicFormat.SubjectPublicKeyInfo
        ))
    return private_key, public_key
```

Slika 12. Generisanje RSA ključeva

```
175     # Sign-then-Encrypt
176     sig = sign_message(text, sender_priv)
177     cipher = encrypt_message(text, recipient_pub)
```

Slika 13. Slanje poruke (potpisivanje i šifrovanje)

```
137     context = ssl._create_unverified_context()
138     credentials = pika.PlainCredentials("test_client", "*****")
139     params = pika.ConnectionParameters(
140         host=RABBITMQ_HOST,
141         port=RABBITMQ_PORT,
142         ssl_options=pika.SSLOptions(context),
143         credentials=credentials
144     )
```

Slika 14. TLS konekcija sa RabbitMQ serverom

```
194     message = decrypt_message(cipher, priv)
195
196     sender_pub_json = requests.get(f"{REGISTRY}/key/{sender}", verify=str(CA_PATH)).json()
197     sender_pub = serialization.load_pem_public_key(sender_pub_json['public_key'].encode())
198     verify_signature(message, sig, sender_pub)
```

Slika 15. Prijem i verifikacija poruke

5. ZAKLJUČAK

U ovom seminarskom radu predstavljena je implementacija bezbedne aplikacije za razmenu poruka zasnovane na RabbitMQ serveru, uz primenu TLS protokola, RSA asimetrične kriptografije i digitalnog potpisivanja poruka. Cilj rada bio je da se prikaže kako se savremeni kriptografski mehanizmi mogu primeniti u realnoj komunikaciji između klijenata, uz obezbeđivanje poverljivosti, integriteta i autentifikacije poruka.

Server ima ulogu registra korisnika i javnih ključeva, dok se kompletna kriptografska obrada poruka obavlja na klijentskoj strani. Ovakva podela odgovornosti doprinosi većoj bezbednosti sistema, jer privatni ključevi nikada ne napuštaju klijentski uređaj. Komunikacija između klijenata i RabbitMQ servera dodatno je zaštićena TLS protokolom, čime se sprečava prisluškivanje i neovlašćen pristup podacima.

Implementirani server može se instalirati i na Raspberry Pi uređaj, što predstavlja praktično rešenje zbog niske cene, male potrošnje energije i dovoljne procesorske snage za ovakav tip aplikacije. Ovakav pristup čini sistem pogodnim za edukativne i eksperimentalne svrhe, kao i za manja zatvorena okruženja. Predstavljena aplikacija može poslužiti kao osnova za dalja unapređenja i proširenja funkcionalnosti u budućem radu.

LITERATURA

- https://www.cloudflare.com/learning/ssl/what-is-tls/
- https://www.cloudflare.com/learning/ssl/what-is-an-ssl-certificate/
- https://www.cloudflare.com/learning/ssl/how-does-ssl-work/
- https://en.wikipedia.org/wiki/Transport_Layer_Security
- https://en.wikipedia.org/wiki/Public-key_cryptography
- https://en.wikipedia.org/wiki/RSA_(cryptosystem)
- https://www.rfc-editor.org/rfc/rfc8446
- https://www.rabbitmq.com/ssl.html
- https://www.rabbitmq.com/tutorials/tutorial-one-python.html
- https://www.rabbitmq.com/documentation.html
- https://pika.readthedocs.io/en/stable/
- https://fastapi.tiangolo.com/
- https://www.raspberrypi.com/products/
- https://cryptography.io/en/latest/
- https://docs.python.org/3/library/ssl.html