IDATT2104 - Datakom – Arbeidskrav 2

Innholdsfortegnelse

[Øving 3 2](#_Toc160052380)

[Oppgaven 2](#_Toc160052381)

[Programkode 2](#_Toc160052382)

[Lagene: 4](#_Toc160052383)

[Klient-tjener 5](#_Toc160052384)

[Three way handshake: 5](#_Toc160052385)

[Dataoverføring 6](#_Toc160052386)

[Tilkoblingen avsluttes 6](#_Toc160052387)

[webserver 6](#_Toc160052388)

[Øving 4 8](#_Toc160052389)

[Oppgaven 8](#_Toc160052390)

[UDP kalkulator 9](#_Toc160052391)

[programkode 9](#_Toc160052392)

[UDP kommunikasjon 12](#_Toc160052393)

[TLS 12](#_Toc160052394)

[Programkode 12](#_Toc160052395)

[Lagene: 15](#_Toc160052396)

[Kommunikasjon 15](#_Toc160052397)

[Nedkobling 15](#_Toc160052398)

[Tjenerens sertifikat 16](#_Toc160052399)

[Kryptografisuite 16](#_Toc160052400)

[Hvordan sesjonsnøkler opprettes 17](#_Toc160052401)

Oppgaven ble utført alene og alt er derfor gjort i localhost. Dette gjør at alle ip-adresser samt at det ikke er tatt i bruk noen mac adresser for overføring av data.  
  
Min lokale IPv4 adresse er: 127.0.0.1

Min mac adresse er: ether f0:18:98:2b:4d:ce

## Øving 3

### Oppgaven

Oppgaven gikk ut på å lage en nettverksapplikasjon basert på klient/tjener-modellen.

I del 1 skulle det utvikles en enkel klient/tjener-applikasjon hvor klienten sender to tall til tjeneren, hvor tjener da skal enten addere eller subtrahere tallene basert på hva bruker ønsker.

Tjeneren utfører deretter den ønskede operasjonen og returnerer resultatet tilbake til klienten. Tjeneren skal også sende tilbake passende feilmelding om feil skulle oppstå.

Interaksjonen skal kunne gjentas slik at flere beregninger kan utføres i løkke.

Videre skal tjenerapplikasjonen utvides til å kunne håndtere flere samtidige klientforbindelser. Dette oppnås ved å implementere multitråding på tjenersiden, hvor hver klientforbindelse betjenes av en egen tråd.

Del 2 av Oppgaven innebærer å lage en enkel webtjener som håndterer én klient om gangen. Når en klient kobler seg til, skal tjeneren returnere en HTML-side med en `<H1>` velkomstmelding og en punktliste (`<UL>` med `<LI>` elementer) som viser klientens HTTP-header.

### Programkode

Applikasjonen er en enkel nettverksapplikasjon laget i C++. Den består av de tre komponentene: Client, Server og Webserver.

Klientkoden din oppretter en TCP-forbindelse til serveren og sender deretter data basert på brukerinput og mottar svar.

**Opprette Socket:** Koden starter med å opprette en socket med «AF\_INET» for IPv4 og TCP-kommunikasjon. Hvis socket ikke kan opprettes, vises en feilmelding, og programmet avslutter.

**Konfigurere Serveradressen:** Den setter opp serverens adresseinformasjon, inkludert IP-adressen (her er det "127.0.0.1" for localhost) og portnummeret (8080). Denne adressen forteller klienten hvor den skal koble til.

**Koble til Serveren:** connect() forsøker å koble til serveren ved hjelp av den tidligere konfigurerte adressen. Ved en feil i tilkoblingsforsøket returneres et negativt tall, som avslutter programmet med en feilmelding.

**Kommunikasjon med Serveren:** Klienten går inn i en løkke hvor den venter på brukerinput. Brukeren kan enten skrive en operasjon eller 'q' for å avslutte. Inputen sendes til serveren, og klienten venter deretter på et svar. Når svaret mottas, vises det for brukeren. Bufferen renses etter hver melding for å forhindre datalekkasje mellom meldingene.

**Avslutte Forbindelsen**: Når brukeren bestemmer seg for å avslutte ved å skrive «q», lukker klienten socketen og avslutter programmet.

Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare

Automatisk generert beskrivelse

Figur 1: Client

Serveren flertrådet TCP-server som lytter på port 8080, aksepterer innkommende forbindelser, og håndterer hver klientforbindelse i sin egen tråd.

**Opprette og Konfigurere Socket:** Serveren oppretter en socket og konfigurerer den for å tillate gjenbruk av adresse/port for å unngå "address already in use" feil. Dette gjøres med «socket()» og «setsockopt()».

**Binde Socket til en Port:** Serveren binder den opprettede socketen til porten slik at den kan lytte etter innkommende forbindelser på denne porten. «bind()» funksjonen brukes til dette formålet.

**Lytte etter Innkommende Forbindelser:** Med «listen()», begynner serveren å lytte på den angitte porten, klar til å akseptere innkommende forbindelser.

**Akseptere Forbindelser og Opprette Tråder:** I en uendelig løkke, venter serveren på nye klientforbindelser med «accept()». For hver ny forbindelse som aksepteres, opprettes en ny tråd ved hjelp av «pthread\_create()». Denne tråden vil håndtere kommunikasjonen med den tilkoblede klienten.

**Håndtering av Klientforbindelser:** «handleClient» funksjonen kjøres i en separat tråd for hver klient. Den leser klientens forespørsel, tolker den, utfører en beregning basert på forespørselen og sender resultatet tilbake til klienten. Hvis klienten sender "q" eller lukker forbindelsen, avsluttes tråden.

Et bilde som inneholder tekst, skjermbilde, programvare

Automatisk generert beskrivelse

Figur 2: Server

### Lagene:

**Transportlaget**

I koden brukes TCP. TCP, som står for Transmission Control Protocol, er en nøkkelkomponent i internettets transportlag og spiller en viktig rolle i å sikre pålitelig, tilkoblingsorientert kommunikasjon mellom enheter på nettet. Ved å bruke TCP, kan applikasjoner som webservere og klienter utveksle data med garantier om at informasjonen ankommer i riktig rekkefølge, uten feil, og på en kontrollert måte som forhindrer overbelastning av nettverket.

Ved å se på bruken av «SOCK\_STREAM» kan en se at det brukes TCP.

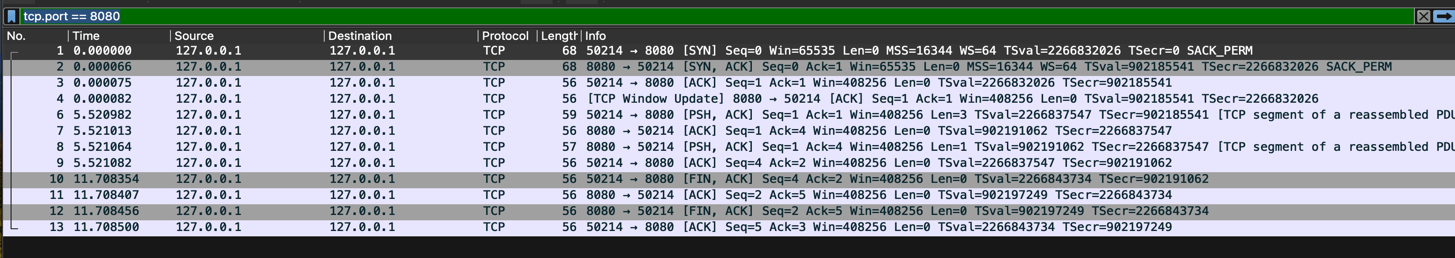
**Nettverkslaget**

Koden bruker også internett protokollen. På nettverkslaget, fungerer Internet Protocol, som ryggraden i dataoverføring over internett. IP sørger for adressering og ruting av datapakker fra sender til mottaker over diverse nettverk. Det gjør det mulig for pakker å navigere gjennom komplekse nettverk ved å bruke IP-adresser, som unikt identifiserer hver enhet på nettet. Selv om IP ikke garanterer pålitelig levering på egen hånd, kombineres det effektivt med TCP for å gi en robust og effektiv ende-til-ende forbindelse.

En kan se at det brukes ipv4 i koden ved å se på «AF\_INET»

### Klient-tjener

Bildet viser kommunikasjon mellom klient og tjener



Figur 3: wireshark - tcp

### Three way handshake:

Three-way handshake brukes i TCP/IP-nettverk for å etablere en pålitelig forbindelse mellom en klient og en server. Den består av tre trinn:

**SYN:** Klienten sender en SYN (synchronize) melding til serveren for å starte en forbindelse og angir en sekvensnummer for å starte sekvenseringen av meldingene.

**SYN-ACK:** Serveren svarer med en SYN-ACK (synchronize-acknowledge) melding. Denne meldingen bekrefter mottak av klientens SYN og sender sitt eget sekvensnummer som en del av handshake-prosessen.

**ACK:** Klienten sender en ACK (acknowledge) melding tilbake for å bekrefte mottak av serverens SYN-ACK melding, igjen ved å øke sekvensnummeret med én.

Etter denne prosessen er forbindelsen etablert, og data kan begynne å overføres mellom klienten og serveren. Three-way handshake sikrer at begge parter er klare for dataoverføring og bidrar til å synkronisere sekvensnumrene som brukes for å holde styr på dataene som sendes.

Et bilde som inneholder tekst, programvare, nummer, Font

Automatisk generert beskrivelse

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nr** | **Source / destination** | **protocol** | **Port** | **Info** | **Seq, Act** |
| 1 | 127.0.0.1 | TCP | 50214 - 8080 | SYN | 0,- |
| 2 | 127.0.0.1 | TCP | 8080 - 50214 | SYN, ACT | 0,1 |
| 3 | 127.0.0.1 | TCP | 50214 - 8080 | ACT | 1,1 |

Pakke Nr 4 (TCP window update) indikerer at serveren har ledig plass i bufferen og kan ta imot mer data fra klienten.

En kan også se «PSH, ACK» dette er regneoperasjonene.

### Dataoverføring

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nr** | **Source / Destinatio n** | **protocol** | **Port** | **Info** | **Seq, Act** |
| 6 | 127.0.0.1 | TCP | 50214 - 8080 | PSH, ACK | 1,1 |
| 7 | 127.0.0.1 | TCP | 8080 - 50214 | ACK | 1,1 |
| 8 | 127.0.0.1 | TCP | 50214 - 8080 | PSH, ACK | 1,4 |
| 9 | 127.0.0.1 | TCP | 8080 - 50214 | ACK | 1,4 |

Pakke 5: Serveren gir data til klienten. PSH-flagget viser at serveren ber operativsystemet om å sende dataen til mottakerapplikasjonen. ACK-flagget anerkjenner den siste mottatte pakken fra klienten.

Pakke 6: Klienten har mottatt dataene som ble sendt i pakke 5. Bekreftelsesnummeret 1 indikerer at klienten har mottatt data opp til byte 1 og forventer byte 1 som neste byte.

### Tilkoblingen avsluttes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nr** | **Source / Destinatio n** | **protocol** | **Port** | **Info** | **Seq, Act** |
| 10 | 127.0.0.1 | TCP | 50214 - 8080 | FIN, ACK | 4,2 |
| 11 | 127.0.0.1 | TCP | 8080 - 50214 | ACK | 2,5 |
| 12 | 127.0.0.1 | TCP | 50214 - 8080 | FIN, ACK | 2,5 |
| 13 | 127.0.0.1 | TCP | 8080 - 50214 | ACK | 5,3 |

Klienten sender til slutt «FIN, ACK». Dette er klientens måte å initialisere termineringen av koblingen. Dette betyr at den er ferdig å sende data

### webserver

Koden starter en enkel webserver som lytter på port 3000. Ved oppstart forsøker den å åpne nettleseren din automatisk til adressen «http://localhost:3000». Serveren oppretter en socket for å lytte etter innkommende forbindelser, konfigurerer den for gjenbruk av adresse/port, og binder den til den angitte porten på alle tilgjengelige nettverksinterfjes.

Når serveren aksepterer en innkommende forbindelse, leser den HTTP-forespørselen fra klienten og bygger en HTTP-respons. Responsen inneholder en statuslinje for å indikere at forespørselen ble vellykket behandlet, en content-type header for å spesifisere at innholdet er HTML, og en enkel HTML-side. HTML-siden viser en velkomstmelding og lister opp hver linje i HTTP-forespørselen som klienten sendte, formatert som en punktliste.

Til slutt sender serveren denne responsen tilbake til klienten, lukker klientforbindelsen, og avslutter ved å lukke sin egen lytte-socket. Denne prosessen demonstrerer hvordan en server kan akseptere en forbindelse, behandle en forespørsel, og sende et svar tilbake til klienten.

Et bilde som inneholder tekst, skjermbilde, programvare, Operativsystem

Automatisk generert beskrivelse

Figur 4: webserver

Et bilde som inneholder tekst, skjermbilde, Font, nummer

Automatisk generert beskrivelse

Figur 5: wireframe – http

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nr** | **Source / destination** | **protocol** | **Port** | **Info** | **Seq, Act** |
| 3 | 127.0.0.1 | TCP | 50537 - 8080 | SYN | 0,- |
| 4 | 127.0.0.1 | TCP | 8080 - 50537 | SYN, ACT | 0,1 |
| 5 | 127.0.0.1 | TCP | 50537 - 8080 | ACT | 1,1 |

**http**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr** | **Source / Destinatio n** | **protocol** | **Info** |  |
| 7 | 127.0.0.1 | HTTP | GET / HTTP/1.1 |  |
| 11 | 127.0.0.1 | HTTP | HTTP/1.1 200 OK |  |

Pakke 7: Inneholder en HTTP GET-forespørsel fra klienten til serveren for å hente hovedsiden «/».

Pakke 11: Dette er serverens HTTP 200 OK-respons på GET-forespørselen i pakke 7, som indikerer at forespørselen var vellykket og at det blir returnert en HTML-fil.

## Øving 4

### Oppgaven

Denne oppgaven består av to deler. I den første delen av oppgaven, skaper du en kalkulatorapplikasjon der en klient sender beregningsforespørsler til en server ved hjelp av UDP, en forbindelsesløs protokoll kjent for sin enkelhet og effektivitet for enkel dataoverføring. Denne delen fokuserer på grunnleggende nettverksprogrammering og er en øvelse i å sende og motta data over nettverk uten en etablert forbindelse.

I den andre delen av oppgaven, dykker du dypere inn i nettverkssikkerhet ved å sette opp en server som bruker TLS/SSL, protokoller designet for å sikre kryptert kommunikasjon over internett.

### UDP kalkulator

programkode  
client:

Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare

Automatisk generert beskrivelseEt bilde som inneholder tekst, programvare, Multimedieprogramvare, Grafikkprogramvare

Automatisk generert beskrivelse

Figur 6: client (p4)

Klienten er en del av kalkulatorapplikasjon som kommuniserer med en server via UDP. Ved oppstart forsøker klienten å opprette en UDP-socket. Hvis dette mislykkes, avsluttes programmet med en feilmelding.

Klienten konfigurerer deretter serverens adresseinformasjon, inkludert IP-adresse og portnummer 9000, for å vite hvor meldingene skal sendes. Ved hjelp av en løkke, lar klienten brukeren kontinuerlig utføre matematiske operasjoner. For hver iterasjon, ber den brukeren om to tall og hvilken matematisk operasjon som skal utføres på disse tallene. Brukeren kan velge mellom addisjon, subtraksjon, multiplikasjon, og divisjon

Når brukeren har oppgitt input, konverteres denne informasjonen til en streng og sendes til serveren ved hjelp av `sendto`-funksjonen. Klienten venter deretter på et svar fra serveren ved å bruke `recvfrom`-funksjonen, som blokkerer inntil et svar mottas eller en feil oppstår. Svaret, som forventes å være resultatet av den matematiske operasjonen, skrives ut til brukeren.

Etter å ha mottatt og vist svaret, spør klienten brukeren om de ønsker å utføre en ny beregning. Hvis brukeren svarer nei, avsluttes løkken, og klienten lukker socketen før programmet avsluttes. Dette tillater brukeren å utføre så mange beregninger som ønsket med serveren før programmet avsluttes på en kontrollert måte.

**Server:**

**Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare

Automatisk generert beskrivelse**

**Et bilde som inneholder tekst, skjermbilde, programvare

Automatisk generert beskrivelse**

Figur 7: server (p4)

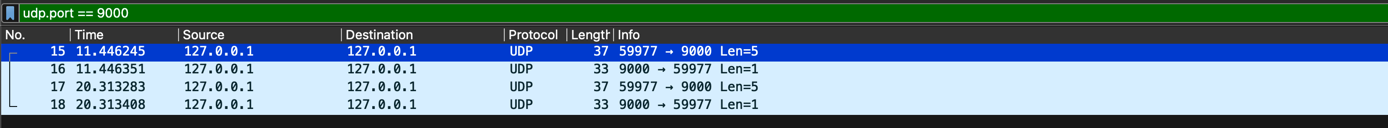
Ved oppstart forsøker serveren å opprette en UDP-socket og binder den til portnummer 9000 for å lytte etter innkommende forespørsler på hvilken som helst interface (`INADDR\_ANY`). Hvis opprettelsen eller bindingen av socketen mislykkes, avsluttes programmet med en feilmelding.

Når serveren er klar og lytter på porten, går den inn i en løkke hvor den venter på forespørsler fra klienter. For hver forespørsel leser serveren meldingen som inneholder to tall og en indikert av symbolene +, -, \*, eller /.

Serveren dekoder meldingen ved hjelp av en `std::istringstream` for å ekstrahere operasjonen og de to tallene. Basert på operasjonen som er forespurt av klienten, utfører serveren den tilsvarende matematiske operasjonen. Hvis operasjonen er gyldig, beregnes resultatet, og serveren genererer en respons som inneholder resultatet av beregningen. For divisjon kontrollerer serveren også om divisor er null for å unngå divisjon med null. Hvis operasjonen ikke er gyldig, eller det er en annen feil, genereres en feilmelding som respons.

Til slutt sender serveren responsen tilbake til klientens adresse ved å bruke `sendto`-funksjonen, som også spesifiserer klientens adresse og portnummer for å sikre at svaret blir sendt tilbake til riktig mottaker. Dette gjøres inne i `handle\_client`-funksjonen, som tar den mottatte meldingen, klientens adresseinformasjon, og serverens socket som argumenter for å behandle forespørselen og sende tilbake et svar.

### UDP kommunikasjon



Figur 8: Wireshark – udp

UDP, som står for User Datagram Protocol, er en enkel, forbindelsesløs nettverksprotokoll som brukes for å sende datagrammer over et IP-nettverk uten å garantere levering, rekkefølge eller feilfri overføring av pakker. Dette gjør UDP til et raskt og effektivt valg for applikasjoner som krever høy ytelse og kan tolerere noe tap av data, som strømming av video eller lyd, spill, eller noen ganger, sanntidskommunikasjon. UDP tillater applikasjoner å sende meldinger, kalt datagrammer, til andre vertsprogrammer på et IP-nettverk med minimal overhead, da det ikke oppretter en forbindelse før data sendes og ikke utfører feilkontroll etter at data er sendt.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr** | **Source / destination** | **protocol** | **Port** | **Info** |
| 15 | 127.0.0.1 | UDP | 59977 - 9000 | LEN = 5 |
| 16 | 127.0.0.1 | UDP | 9000 - 59977 | LEN = 1 |
| 17 | 127.0.0.1 | UDP | 59977 - 9000 | LEN = 5 |
| 18 | 127.0.0.1 | UDP | 9000 - 59977 | LEN = 1 |

### TLS

Programkode

**JavaSSLServer**:  
  
koden implementerer en enkel SSL-sikret server som lytter etter innkommende tilkoblinger på port 8000. Serveren bruker Java sin `SSLServerSocketFactory` for å opprette en SSL-beskyttet `ServerSocket`, som sikrer at all data som sendes mellom serveren og klientene er kryptert og sikker.

Når serveren er startet, skriver den ut en melding om at den lytter etter tilkoblinger, og venter deretter på at en klient skal koble seg til. Når en klient kobler seg til serveren, aksepterer den tilkoblingen og oppretter en `Socket` for å kommunisere med klienten.

For å håndtere dataoverføring, oppretter serveren en `PrintWriter` og en `BufferedReader`. `PrintWriter` brukes til å sende data tilbake til klienten, mens `BufferedReader` leser data som kommer fra klienten. Serveren leser linjer med tekst fra klienten og skriver ut disse linjene til konsollen. Deretter sender den samme linjen tilbake til klienten som en ekko-respons.

Kommunikasjonen fortsetter i en løkke hvor serveren leser og ekkoer linjer med tekst til den ikke mottar flere data, hvorpå den avslutter tilkoblingen. Til slutt, skriver serveren ut en melding om at forbindelsen er lukket.

**Et bilde som inneholder tekst, skjermbilde

Automatisk generert beskrivelseEt bilde som inneholder tekst, skjermbilde

Automatisk generert beskrivelse**

Figur 9: javaSSLServer

**JavaSSLClient:**

Koden implementerer en enkel SSL-klient som kobler seg til en server på port 8000 ved hjelp av SSL for en sikret kommunikasjon. Klienten bruker `SSLSocketFactory` for å opprette en SSL-sikret `Socket` forbindelse til serveren, som sikrer at data som utveksles mellom klienten og serveren er kryptert og beskyttet mot avlytting eller datainnblanding.

Når tilkoblingen er etablert, oppretter klienten en `PrintWriter` for å sende data til serveren og en `BufferedReader` for å motta data fra serveren. Programmet går deretter inn i en løkke hvor det ber brukeren om å skrive inn tekst via konsollen. Denne teksten sendes til serveren, og klienten venter på et svar. Når svaret mottas, vises det for brukeren. Dette fortsetter til brukeren skriver inn "q", som er signalet for å avslutte løkken og dermed også programmet.

****

Figur 10: javaSSLClient

### Lagene:

I kommunikasjonen mellom SSL/TLS-sikrede servere og klienter benyttes flere lag av nettverksprotokoller for å sikre en pålitelig og sikker dataoverføring. På transportlaget, opererer SSL eller TLS for å legge til et lag av sikkerhet ved å kryptere dataene som sendes over nettverket.

På nettverkslaget benyttes IP for å adressere og rute pakker fra sender til mottaker.

### Kommunikasjon

Et bilde som inneholder tekst, skjermbilde, programvare

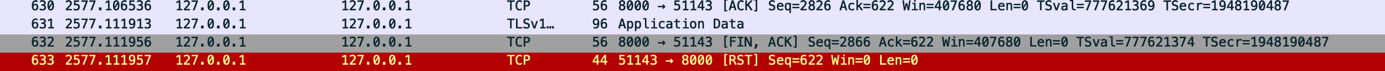
Automatisk generert beskrivelse

Figur 11: wireshark ssl

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nr** | **Source / destination** | **protocol** | **Port** | **Info** | **Seq, Act** |
| 466 | 127.0.0.1 | TCP | 51143 - 8000 | SYN | 0,- |
| 467 | 127.0.0.1 | TCP | 8000 - 51143 | SYN, ACT | 0,1 |
| 468 | 127.0.0.1 | TCP | 51143 - 8000 | ACT | 1,1 |

Etter tilkoblingen med three way handshake blir det sendt «hello client», deretter «hello server». Med dette foreslår klienten SSL/TLS protokoller og “cipher suites”. Med Server Hello velger serveren hvilke protokoller og cipher suite som faktisk skal bli brukt.  
  
En kan også se det står «Change Cipher Spec». Dette er en melding brukt i protokollene for å indikere at partene i en kommunikasjon nå vil begynne å bruke de avtalte sikkerhetsinnstillingene for kryptering av meldinger. Denne meldingen sendes som en del av håndtrykksprosessen for å signalisere overgangen fra usikret til sikret kommunikasjon ved å aktivere den valgte krypteringsalgoritmen.

### Nedkobling



Figur 12: nedkobling av ssl

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nr** | **Source / destination** | **protocol** | **Port** | **Info** | **Seq, Act** |
| 630 | 127.0.0.1 | TCP | 8000 - 51143 | SYN | 2826,622 |
| 631 | 127.0.0.1 | TLSv1.3 |  | SYN, ACT |  |
| 632 | 127.0.0.1 | TCP | 8000 - 51143 | ACT | 22866,62 |
| 633 | 127.0.0.1 | TCP | 51143 - 8000 | RST | 622 |

|  |  |
| --- | --- |
| TLS- versjon | TLSV1.3(version 1.2) |
| Hvilken kryptografisuite som velges | TLS\_AES\_256\_GCM\_SHA384 |

### Tjenerens sertifikat

Sertifikatet under har et unikt fingeravtrykk og signatur som er unikt for dette sertifikatet.

Når du bruker kommandoen `keytool -list -v -keystore examplestore`, listes det opp alle nøklene og sertifikatene som er lagret, inkludert informasjon som alias for hver oppføring, sertifikatets eier og utsteder, serienummer, gyldighetsperiode, og den offentlige nøkkelen.

Sertifikatet bruker en 2048-bits RSA nøkkel

Et bilde som inneholder tekst, skjermbilde, Font

Automatisk generert beskrivelse

### Kryptografisuite

krypteringssuiten som brukes er «TLS\_AES\_256\_GCM\_SHA384»

Krypteringsuiten "TLS\_AES\_256\_GCM\_SHA384" beskriver en sett med algoritmer som brukes for å sikre kommunikasjon i en TLS (Transport Layer Security) sesjon. Den gir detaljert informasjon om hvilke krypterings- og hashalgoritmer som anvendes for å beskytte dataene som utveksles.

AES\_256\_GCM: Dette står for Advanced Encryption Standard med en nøkkellengde på 256 bit, som brukes for kryptering av meldingsinnholdet. GCM er en modus for operasjonen som tilbyr både autentisering og konfidensialitet (kryptering), og er kjent for sin effektivitet og sikkerhet.

SHA384: Dette refererer til bruk av SHA-384 for å generere en meldingsautentiseringskode Dette sikrer integriteten og autentisiteten til dataene som sendes, ved å beskytte mot uautoriserte endringer.

Samlet sett indikerer "TLS\_AES\_256\_GCM\_SHA384" at krypteringsuiten bruker AES med en 256-bits nøkkel i GCM-modus for å kryptere data, og SHA-384 for å sikre integriteten og autentisiteten til meldingene.

### Hvordan sesjonsnøkler opprettes

Sesjonsnøkler opprettes gjennom en prosess kjent som nøkkelutveksling, som er en del av håndtrykket mellom to parter i en kryptert kommunikasjon, for eksempel mellom en nettleser og en webserver over TLS. Denne prosessen starter når partene først blir enige om en krypteringsalgoritme og deretter bruker en sikker metode for å dele eller etablere en hemmelig nøkkel. Under nøkkelutvekslingen genereres offentlige og private nøkler; den private nøkkelen holdes hemmelig, mens den offentlige nøkkelen deles med den andre parten. Ved hjelp av disse nøklene kan partene beregne en delt hemmelighet uten å faktisk overføre den over nettverket. Denne delte hemmeligheten, sammen med tilleggsinformasjon som er unik for hver sesjon, brukes deretter til å generere sesjonsnøklene. Sesjonsnøklene er midlertidige og brukes til å kryptere og dekryptere data som sendes i den aktuelle kommunikasjonssesjonen, sikrende at overført informasjon forblir konfidensiell og uforandret under overføringen.