IDATT2104 DKO - Oblig 1

Gruppesamarbeid: Karl Labrador, Lars-Håvard Holter Bråten

Oppgave 1 - Wireshark og DNS

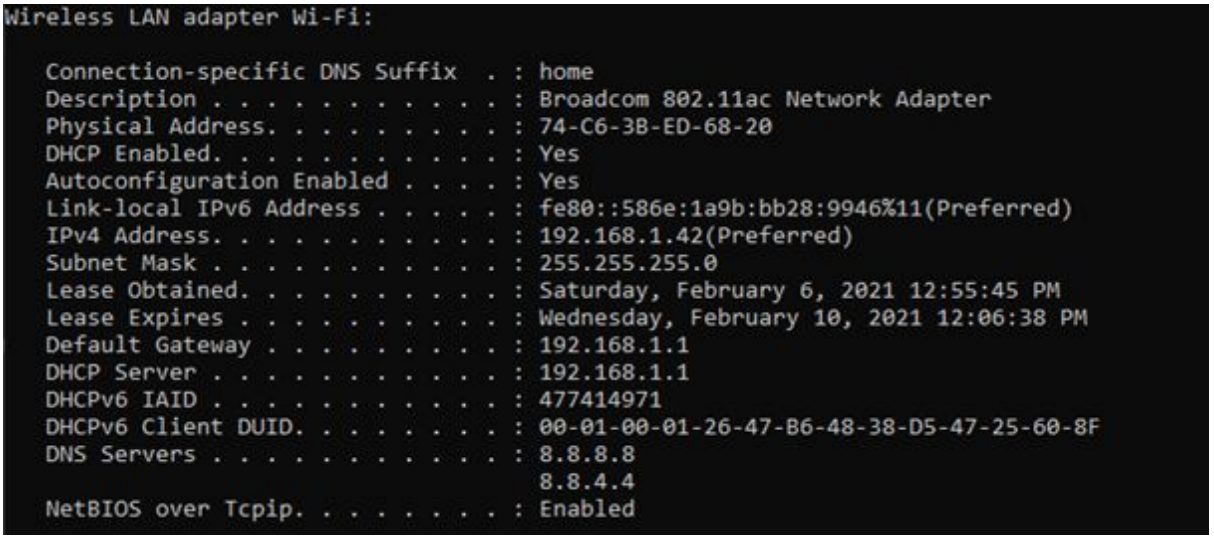
Oppgave 1A

Skjermbilde:

Ved å bruke “nslookup” kan man gjøre DNS-oppslag for å hente IP-adresser til forskjellige domenenavn. DNS-tjenesten oversetter domenenavn om til IP-adresser, og motsatt. Her (i skjermbildet) brukes Google sin DNS-tjeneste (8.8.8.8) for å gjøre oppslag.

“Non-authorative answer” betyr at svaret fra DNS-oppslaget ikke er hentet direkte fra Google sin DNS, men at Google sin DNS-tjeneste har sendt forespørselen videre til en navneserver som har svaret. Dersom man hadde gjort navneoppslag mot datakom.no sin SOA (Start of Authority), eventuelt NS (Name Server), så ville man fått et “Authorative answer”.

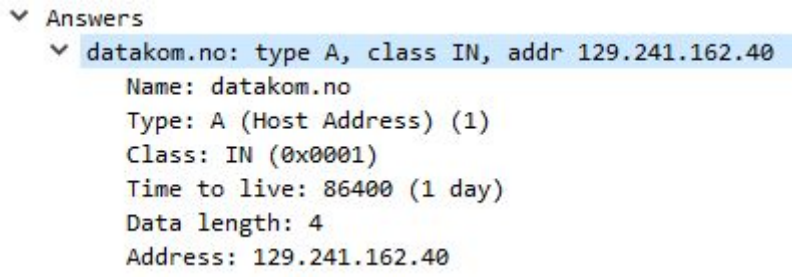
Oppgave 1B



IP-adressen til DNS i ipconfig /all er den samme som blir brukt i nslookup, med mindre man spesifiserer hvilken DNS-server man vil bruke i nslookup.

Oppgave 1C

Skjermbilde:



Her ser vi at type er A, og levetiden er 86400 sekunder (1 dag). Typen viser at informasjonen vi har fått er en IPv4-adresse. Levetiden forteller klienten hvor lenge vi skal lagre informasjonen.

Oppgave 1D

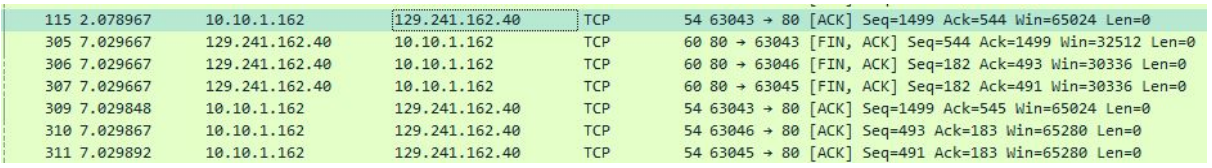
I tillegg til DNS-oppslag av type A gjør den også oppslag av type AAAA. Typen AAAA er IPv6-adresse.

Oppgave 2 - Wireshark og HTTP

Oppgave 2A

Klienten sender headerlinja “Connection: keep-alive” i forespørselen. Tjeneren svarer med “Keep-Alive: timeout=5, max=100”.

Skjermbilde fra Wireshark:



Her kan man se at det går ca. 5 sekunder (tilnærmet lik timeout) fra siste svar fra tjeneren til det kommer en TCP-pakke der FIN-flagget er satt.

Oppgave 2B

Klienten sender headerlinjene “Cache-Control: max-age=0” og “If-Modified-Since: viss dato” i forespørselen. Klienten ber tjeneren overføre innholdet på nytt dersom innholdet er endret etter den datoen.

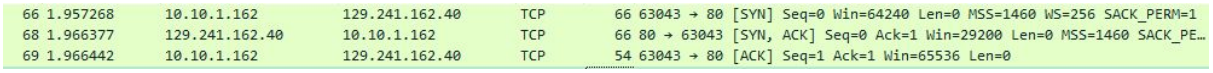
Tjeneren svarer enten ved å sende statuskode 200 OK med nytt innhold, eller statuskode 304 Not Modified. Dersom klienten mottar statuskode 304 vil den bruke innholdet som er lagret i lokalt mellomlager.

Oppgave 3 - Wireshark og TCP

Oppgave 3A - Oppkopling 3-way handshake

SRC		DST	FLAGG	SEQNR	ACKNR
63043	➔	80	SYN	0	0
80	➔	63043	SYN, ACK	0	1
63043	➔	80	ACK	1	1

Skjermbilde:



Sekvensnummeret begynner alltid på 1, og tjeneren mottar 1 byte.
Kvitteringsnummeret som blir sendt tilbake av tjener blir summen av mottatte pakkes sekvensnummer pluss lengden av nyttelasten som ble mottatt. Acknummeret blir derfor 2.

Oppgave 3B - Dataoverføring

NR	Avsender	DST-port	SEQNR	ACKNR	TCP PAYLOAD	Merknad
1	Klient	80	1	1	447	GET-request to webtjener
2	Webtjener	61982	1	448	1460	TCP segment
3	Webtjener	61982	1461	448	1460	TCP segment
4	Webtjener	61982	2921	448	1460	TCP segment
5	Webtjener	61982	4381	448	1460	TCP segment
6	Webtjener	61982	5841	448	1117	HTTP/1.1 200 OK (text/html)
7	Klient	80	448	6958	338	GET /boker.css HTTP/1.1
8	Webtjener	61982	6958	786	1460	TCP segment
9	Webtjener	61982	8418	786	759	HTTP/1.1 200 OK Text/css
10	Klient	80	786	9177	411	GET /bitmaps/forlop_datakommunikasjon.jpg HTTP/1.1
11	Webtjener	61982	9177	1197	1460	TCP segment
12	Webtjener	61982	10637	1197	1140	HTTP/1.1 200 OK JPEG JFIF image

13	Klient	61982	1197	11777	386	GET /favicon.ico HTTP/1.1
14	Webtjener	61982	11777	1583	488	HTTP/1.1 404 Not found (text/html)

Oppgave 4 - Nettverk subnetting

For å få til en “lovlig” oppdeling av nettene har vi måttet splitte 24 bitnettet flere ganger slik at vi endte opp med fem nett, hvorav to er foreløpig ubrukte. Nettene kunne også blitt splittet opp slik at slik at vi hadde endt opp med et 25 bit nett og to 26 bit nett. Vi ville da endt opp med tre større nett, noe som i våre øyne hadde vært uryddig og overkill i forhold til behovet på for eksempel tjenernettet.

Nett	**Nett-adresse**	**CIDR**	**Min host**	**Gateway**	**Max host**	**Broadcast**	**Tilgjengelige**	**Behov**	**Ledig**
C Tjenernett	a.b.c.0	/28	a.b.c.1	a.b.c.1	a.b.c.14	a.b.c.15	14	10	3
IKKE I BRUK	a.b.c.16	/28	a.b.c.17	a.b.c.17	a.b.c.30	a.b.c.31	14	Null	13
B Gjestenett	a.b.c.32	/27	a.b.c.33	a.b.c.33	a.b.c.62	a.b.c..63	30	20	9
IKKE I BRUK	a.b.c.64	/26	a.b.c.65	a.b.c.65	a.b.c.126	a.b.c.127	62	Null	61
A Ansattnett	a.b.c.128	/25	a.b.c.129	a.b.c.129	a.b.c.254	a.b.c.255	126	80	45

Oppgave 5 - Dok. av Øving 1 - Tråder og Socketprogrammering

Øvingens del 1: Enkel tjener/klient (kalkulator)

Aktuell kode på serversiden:

```
PrintWriter printer = new PrintWriter(socket.getOutputStream(), autoFlush: true);

// Welcome message
printer.println("You are now connected to the server. Welcome!");
printer.println("Commands: add, sub, exit");
```

I skjermbildet ovenfor ser man aktuell kode for å sende meldinger til klienten. Her etableres det en outputstream som man kan skrive informasjon til. Når den flushes (som skjer automatisk i dette tilfellet), sendes informasjonen til klienten på socketen.

Motparten til aktuell kode, men på klientsiden:

```
InputStreamReader inputReader = new InputStreamReader(socket.getInputStream());
BufferedReader reader = new BufferedReader(inputReader);
PrintWriter printer = new PrintWriter(socket.getOutputStream(), autoFlush: true);

// Welcome message
System.out.printf("%s%n%s%n", reader.readLine(), reader.readLine());
```

Her etableres det en inputstream som leser informasjon som blir sendt på samme socket som både klient og server er tilknyttet.

Meldingsutvekslingen som skjer i kodesnuttene er når en klient kobler seg til serveren. Skjermbilde fra Wireshark:

301	12.110914	127.0.0.1	127.0.0.1	TCP	56 59528 → 1250 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
302	12.110947	127.0.0.1	127.0.0.1	TCP	56 1250 → 59528 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 S
303	12.110966	127.0.0.1	127.0.0.1	TCP	44 59528 → 1250 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
304	12.119926	127.0.0.1	127.0.0.1	TCP	91 1250 → 59528 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=47
305	12.119950	127.0.0.1	127.0.0.1	TCP	44 59528 → 1250 [ACK] Seq=1 Ack=48 Win=2619648 Len=0
306	12.119977	127.0.0.1	127.0.0.1	TCP	70 1250 → 59528 [PSH, ACK] Seq=48 Ack=1 Win=2619648 Len=26
307	12.119983	127.0.0.1	127.0.0.1	TCP	44 59528 → 1250 [ACK] Seq=1 Ack=74 Win=2619648 Len=0

Informasjonen vises i Wireshark i klartekst (her er det kjørt noen kommandoer i tillegg for testing):

Wireshark · Follow TCP Stream (tcp.stream eq 16) · Adapter for loopback traffic capture

```
You are now connected to the server. Welcome!
Commands: add, sub, exit
add 1+1
You must provide two numbers. Example: add 15 10
add 1 1
[add] 1 + 1 = 2
```

Øvingens del 2: Enkel webtjener (returnere HTTP-header)

Oppgave A:

```
// I/O setup
InputStreamReader inputReader = new InputStreamReader(socket.getInputStream());
BufferedReader reader = new BufferedReader(inputReader);
PrintWriter printer = new PrintWriter(socket.getOutputStream(), autoFlush: true);
```

I skjermbildet er det kode for å sette opp input og output streams for utveksling av informasjon. Inputstreamen brukes for å lese data fra nettleseren, slik som request headeren. Outputstreamen brukes for å sende data til nettleseren.

```
// Get & Save Client Header
List<String> header = new ArrayList<>();
String line = reader.readLine();
while (!line.equals("")) {
    header.add(line);
    line = reader.readLine();
}
```

I kodesnutten ovenfor leses headeren i forespørselen fra nettleseren, og lagres for å sende tilbake i HTML-koden senere.

```
printer.println("HTTP/1.0 200 OK");
printer.println("Content-Type: text/html; charset=utf-8");
printer.println("");

// Reply to the request with content
printer.println("<!DOCTYPE html>");
printer.println("<html>");
```

Videre sendes det informasjon tilbake til nettleseren med header og body.

Meldingsinnholdet i klartekst fra Follow TCP-stream i Wireshark:

Wireshark · Follow TCP Stream (tcp.stream eq 5) · Adapter for loopback traffic capture

```
GET / HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:86.0) Gecko/20100101 Firefox/86.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: nb,no;q=0.9,nb-NO;q=0.8,no-NO;q=0.6,nn-NO;q=0.5,nn;q=0.4,en-US;q=0.3,en;q=0.1
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1

HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>
<html>
<head><title>Oblig 1 (Webserver)</title></head>
<body>
<h1>Welcome!</h1>
<p>Your client header is:</p>
<ul>
<li>GET / HTTP/1.1</li><li>Host: localhost</li><li>User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64; rv:86.0) Gecko/20100101 Firefox/86.0</li><li>Accept: text/html,application/
xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8</li><li>Accept-Language:
nb,no;q=0.9,nb-NO;q=0.8,no-NO;q=0.6,nn-NO;q=0.5,nn;q=0.4,en-US;q=0.3,en;q=0.1</li><li>Accept-
Encoding: gzip, deflate</li><li>Connection: keep-alive</li><li>Upgrade-Insecure-Requests: 1</li></
ul>
</body></html>
```

Tjeneren ber om kvittering på mottatt innhold. Klienten kvitterer for mottatt innhold (hver gang outputstreamen på tjeneren flushes).

Oppgave B:

Når et HTTP-svar er sendt til nettlesere, avsluttes forbindelsen. Dette gjøres ved å lukke input- og outputstreamene som er tilknyttet socketen, i tillegg til socketen selv. Se skjermbilde under for aktuell kode:

```
reader.close();
printer.close();
socket.close();
```

Skjermbilde fra Wireshark:

```
138 1.395410    127.0.0.1    127.0.0.1    TCP    44 80 → 59781 [FIN, ACK] Seq=712 Ack=431 Win=2619648 Len=0[Reassembly…
139 1.395419    127.0.0.1    127.0.0.1    TCP    44 59781 → 80 [ACK] Seq=431 Ack=713 Win=2618880 Len=0
145 1.395588    127.0.0.1    127.0.0.1    TCP    44 59781 → 80 [FIN, ACK] Seq=431 Ack=713 Win=2618880 Len=0
147 1.395600    127.0.0.1    127.0.0.1    TCP    44 80 → 59781 [ACK] Seq=713 Ack=432 Win=2619648 Len=0
```

Oppgave 6 - Dok. av Øving 2 - UDP, TLS og ASIO

Øvingens del 1: Kalkulator med UDP

Skjermbilder med aktuell kode for kommunikasjon:

```
InetAddress address = packet.getAddress();
int port = packet.getPort();
String received = new String(packet.getData(),  offset: 0, packet.getLength());

System.out.printf("%s:%s sent: %s%n", address.getHostAddress(), port, received);
```

Her er koden i en while-løkke og venter på å motta pakker som skal behandles.

```
public static void send(String text, InetAddress address, int port) {
    try {
        buf = text.getBytes();
        DatagramPacket packet = new DatagramPacket(buf, buf.length, address, port);
        socket.send(packet);

        System.out.println(new String(packet.getData(),  offset: 0, packet.getLength()));

        System.out.printf("Server sent to %s:%s -> %s%n", address.getHostAddress(), port, text);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

Både server og klient har en tilnærmet lik “send” metode for å forberede og sende pakker over UDP-protokollen til hverandre.

I Wireshark kan man se pakkene som forekommer av en enkelt kalkulasjon over UDP:

1896	31.479948	127.0.0.1	127.0.0.1	UDP	39	54968 → 1260 Len=7
1897	31.480803	127.0.0.1	127.0.0.1	UDP	47	1260 → 54968 Len=15

Sammenlignet med øving 1 der utvekslingen skjer over TCP-protokollen, forekommer det mindre pakker for å kommunisere over UDP. Skjermbilde:

5569…	6552.892114	127.0.0.1	127.0.0.1	TCP	53	57023 → 1250 [PSH, ACK] Seq=1 Ack=74 Win=2619648 Len=9
5569…	6552.892134	127.0.0.1	127.0.0.1	TCP	44	1250 → 57023 [ACK] Seq=74 Ack=10 Win=2619648 Len=0
5569…	6552.892882	127.0.0.1	127.0.0.1	TCP	61	1250 → 57023 [PSH, ACK] Seq=74 Ack=10 Win=2619648 Len=17
5569…	6552.892894	127.0.0.1	127.0.0.1	TCP	44	57023 → 1250 [ACK] Seq=10 Ack=91 Win=2619648 Len=0

Her ser man at utveksling over TCP skjer ved at pakkene sendes med PSH og ACK-flaggene, der serveren må sende kvittering på mottatt melding. Samme skjer også når serveren skal sende et svar til klienten.

Med UDP sendes meldingen til serveren uten måtte sende kvittering tilbake, og serveren sender svar til klienten uten å kreve kvittering på mottatt svar.

Fordelen med TCP her er at protokollen sikrer at utvekslingen ikke inneholder feil/korrupte pakker, og sørger for at pakker med feil sendes på nytt. I tilfellet med en enkel kalkulator, vil bruk av TCP selv sørge for forutsigbar oppførsel i motsetning til UDP med mindre man implementerer noe form for feilsjekk i programkoden (for eksempel, dersom man ikke har fått svar innen en viss tid, så kan man forsøke å sende ny melding).

UDP gir raskere utveksling, men uten funksjonalitet for å sjekke feil i pakkene. Protokollen forkaster pakker med feil. I tillegg er det lavere ressursbruk på server i motsetning til TCP der den må holde tilkoblingene åpne (og i flere tråder med flere brukere).

Øvingens del 2: Etablere sikker kommunikasjon med TLS

Oppgave A

I øvingen skulle vi generere en keystore med verktøyet keytool:
“keytool -genkey -keyalg RSA -alias signFiles -keystore C:\Java\keystores\examplestore”

I dette tilfellet lagres keystore i filen C:\Java\keystores\examplestore. Keytool ser vanligvis i mappa C:\Users\user\.keystores.

Deretter kjøres: “java -jar -Djavax.net.ssl.keyStore=C:\Java\Keystores\examplestore -Djavax.net.ssl.keyStorePassword=password JavaSSLServer.jar” for å starte serveren. Her angir vi keystore som skal brukes.

For klienten gjør det samme, men her angis det truststore: “java -jar -Djavax.net.ssl.trustStore=C:\Java\Keystores\examplestore -Djavax.net.ssl.trustStorePassword=password JavaSSLClient.jar”

Oppgave B

Skjermbilde:

317	16.421030	127.0.0.1	127.0.0.1	TLSv1.2	411 Client Hello
318	16.421054	127.0.0.1	127.0.0.1	TCP	44 8000 → 59693 [ACK] Seq=1 Ack=368 Win=2619648 Len=0
319	16.453221	127.0.0.1	127.0.0.1	TLSv1.3	171 Server Hello
320	16.453245	127.0.0.1	127.0.0.1	TCP	44 59693 → 8000 [ACK] Seq=368 Ack=128 Win=2619648 Len=0
321	16.458230	127.0.0.1	127.0.0.1	TLSv1.3	50 Change Cipher Spec
322	16.458254	127.0.0.1	127.0.0.1	TCP	44 59693 → 8000 [ACK] Seq=368 Ack=134 Win=2619648 Len=0
323	16.461456	127.0.0.1	127.0.0.1	TLSv1.3	50 Change Cipher Spec
324	16.461475	127.0.0.1	127.0.0.1	TCP	44 8000 → 59693 [ACK] Seq=134 Ack=374 Win=2619648 Len=0
325	16.464859	127.0.0.1	127.0.0.1	TLSv1.3	114 Application Data
326	16.464879	127.0.0.1	127.0.0.1	TCP	44 59693 → 8000 [ACK] Seq=374 Ack=204 Win=2619392 Len=0
327	16.466669	127.0.0.1	127.0.0.1	TLSv1.3	986 Application Data
328	16.466683	127.0.0.1	127.0.0.1	TCP	44 59693 → 8000 [ACK] Seq=374 Ack=1146 Win=2618624 Len=0

Klienten tilbyr en liste på 45 siffersuiter i Client Hello. Tjeneren svarer i Server Hello med siffersuiten TLS_AES_128_GCM_SHA256, som er en av de klienten har tilbudt. Deretter sender klienten melding om at den vil bruke en felles siffersuite (“Change Cipher Spec”), og serveren svarer med kvittering.

Oppgave C - Steg 1 (Client Hello)

I denne oppgaven kjøres det oppkobling mot ntnu.blackboard.com. Klienten tilbyr tjeneren disse siffersuitene å velge blant:

```
Cipher Suites (18 suites)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
    Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
```

Oppgave D - Steg 2A (Server Hello)

Tjeneren svarer med valg av siffersuite:

```
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
```

Denne siffersuiten er med i lista fra klienten (markert rød pil).

Oppgave E - Steg 2B (Server Hello, sertifikat)

Etterspurt innhold i serversertifikatet:

- Issuer/Common Name: Amazon
- Subject/Common Name: *.blackboard.com

Skjermbilder:

```
▼ issuer: rdnSequence (0)
  ▼ rdnSequence: 4 items (id-at-commonName=Amazon,id-at-organizationalUnitName
    › RDNSequence item: 1 item (id-at-countryName=US)
    › RDNSequence item: 1 item (id-at-organizationName=Amazon)
    › RDNSequence item: 1 item (id-at-organizationalUnitName=Server CA 1B)
    › RDNSequence item: 1 item (id-at-commonName=Amazon)

▼ subject: rdnSequence (0)
  ▼ rdnSequence: 1 item (id-at-commonName=*.blackboard.com)
    ▼ RDNSequence item: 1 item (id-at-commonName=*.blackboard.com)
      › RelativeDistinguishedName item (id-at-commonName=*.blackboard.com)
```

Oppgave F - Steg 3 (Nøkkelutveksling)

Client Key Exchange:

Server Key Exchange: