

Dataintegritet

Af B208



TITEL:

Dataintegritet

TEMA:

Modeller og virkelighed

PROJEKT PERIODE:

Datalogi 1. semester

1. Sep. - 19. Sep., 2003

GRUPPE:

B208

DELTAGERE:

Johnny Jacobsen

Rune Zimmer Jensen

Anders Kvist

Rune Mosbæk

Jesper Rosenkilde

Thavarajah Sabanathan

Martin Toft

VEJLEDER:

Janne Skyt

SYNOPSIS:

Dette projekt har omhandler forskellige former for dataintegritet. Der er blevet set på hvor i hverdagen dataintegritet er anvendt, samt hvor ofte det bliver brugt uden at brugerne er klar over det. Af denne grund er der blevet lagt vægt på, hvor vigtigt dataintegritet virkeligt er.

Rapporten omhandler hovedsageligt checksums algoritmer. Disse bliver brugt på computeren til kontrol om filer er de samme, her kan nævnes CRC og MD5 checksums algoritmerne.

OPLAGSTAL: 13

SIDEANTAL: 18

BILAGANTAL OG -ART: INGEN

AFSLUTTET DEN: 19. September 2003.

UNDERSKRIFTER:

Forord

Denne rapport er udarbejdet af gruppe B208, som P0-rapport på den Tekniske-Naturvidenskabelige Basisuddannelse på AAU. Målgruppen til denne rapport ligger blandt basister på den Teknisk-Naturvidenskabelige Basisuddannelse. Formålet med projektet er at introducere læseren til begrebet dataintegritet og gennem foranalyse og problemformulering lægge op til et eventuelt efterfølgende P1-projekt.

Rapporten bygger overordnet på:

- CRC algoritmen
- Introduktion til MD5
- Sikkerhed og fejlfinding i algoritmerne

Vi har hovedsageligt brugt websites som kilder. Kildeangivelsen er at finde sidst i rapporten.

Indhold

1	Introduktion	5
1.1	Baggrund for brugen af dataintegritet	5
1.2	Dataintegritet i kontekst	5
1.3	Forskellige metoder til at sikre integriteten af ens data	7
1.4	Troværdighed til integriteten af ens data / sikkerheden af de forskellige metoder .	7
1.5	Brugen af dataintegritet til genskabelse af data	8
2	Fejl ved transport af data	9
2.1	Enkelt bit fejl (Single bit error)	9
2.2	Burst fejl	9
3	Algoritmer	11
3.1	CRC (Cyclic Redundancy Check)	11
3.2	Polynomier (Polynomials)	12
3.3	MD5 (Message Digest v5)	13
3.4	Forskellig brug mellem MD5 og CRC	13
4	Konklusion	15
5	Problemformulering	16
5.1	Det initierende problem	16
5.2	Problemafgrænsing	16
5.3	Problemformulering	16
6	Kildeliste	17
6.1	Bøger	17
6.2	Internetadresser	17

Kapitel 1

Introduktion

1.1 Baggrund for brugen af dataintegritet

De elektroniske medier har udbredt sig meget gennem de sidste årtier, og bliver brugt til arbejde såvel som fornøjelse. Der vil i begge tilfælde være brug for at kunne overføre data, hvor man kan sikre integriteten af det sendte data. Data kan blive beskadiget under forsendelsen, derudover foreligger også risiko for at dataen, der er sendt mellem afsender og modtager, er blevet ændret af tredje part. Begge tilfælde kræver selvfølgelig en metode til at sikre at dataen forbliver uændret.

1.2 Dataintegritet i kontekst

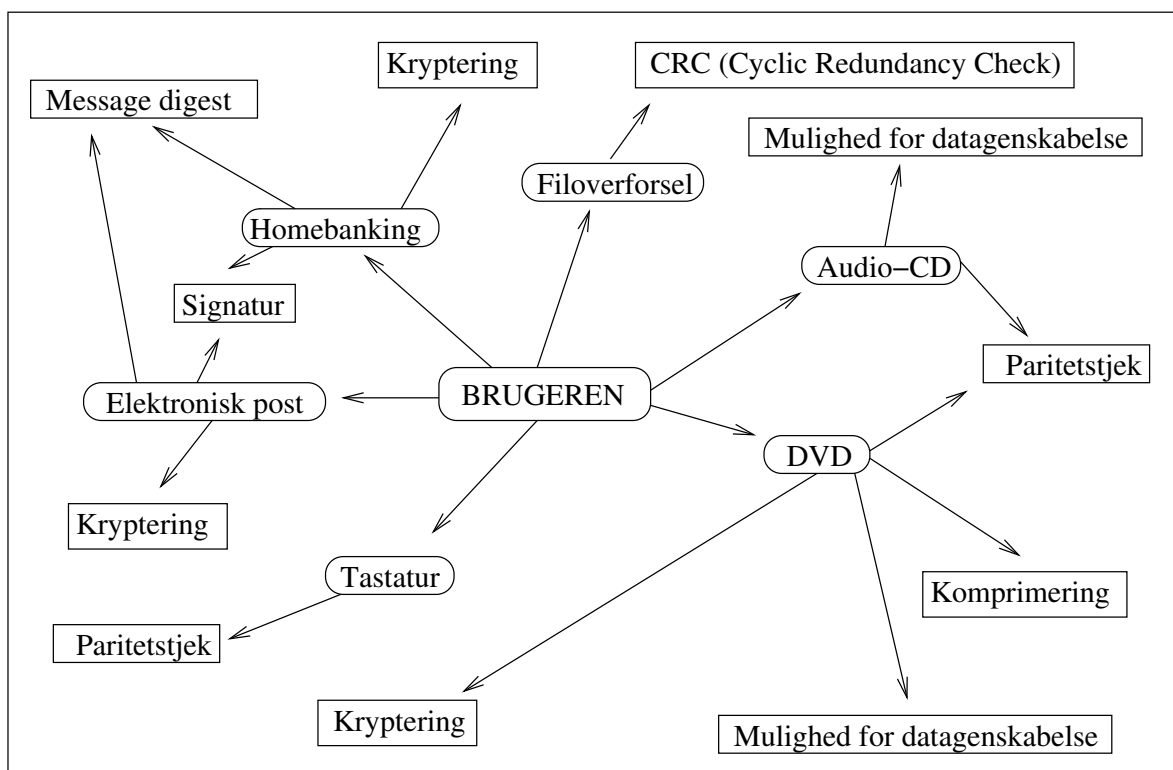
Det er relevant at tage dataintegritet i betragtning ved flere forskellige former for datatransmission og -lagring. En person, der ikke beskæftiger sig med teknikken bag dataintegritet, vil naturligvis ikke have stor viden om emnet. Et flertal af mennesker i vort samfund forventer jo bare, at data overføres og gemmes uden fejl eller besværligheder. I det øjeblik, der sker fx en transmissionsfejl, er disse mennesker, uden at vide det, meget glade for at der på det tekniske plan på forhånd er blevet taget højde for dataintegritet. Med andre ord er dataintegritet ikke noget, man snakker om eller bekymrer sig om, det er noget, der virker transparent for brugeren og forventes at virke problemfrit til alle tider.

Brugeren benytter sig af integritetsbevarende teknologi til en hel række gøremål. Dataintegritet spiller en betydningsfuld rolle for bl.a. homebanking, elektronisk post og diverse medier, se endvidere figur 1.1. Der er dog en mindre forskel på anvendelsen af teknologien - en applikation som fx homebanking stiller store krav til integriteten af overførte informationer samt beskyttelse af disse mod læsning af en evt. mellemmand. En filoverførsels vigtigste mål er naturligvis også at bevare integriteten, dog skal opgaven helst udføres med høj hastighed, hvilket sker på bekostning af de overførte informationers sikkerhed. Det ser for øjeblikket ud til at udviklingen bevæger sig imod, at generel dataudveksling på netværk også skal være sikker mod læsning af mellemmand, hvilket ses på udpræget indførsel af bl.a. VPN (Virtual Private Network) og SFTP (Secure File

Transfer Protocol) over SSH (Secure Shell) i stedet for almindelig FTP, der dog stadig er standard inventar hos mange Internetudbydere.

Scenariet, hvor en bruger vil hente et stykke Open Source software fra et distributionsmirror på Internettet, illustrerer meget godt værdien af dataintegritet. Softwaren kan være alt lige fra kernen i personens operativsystem til et mindre program, pointen er, at det er vigtigt, at dataoverførslen fra afsender til modtager sker korrekt og uden indgriben, da personen naturligvis ønsker, at softwaren virker efter hensigten og ikke indeholder bagdøre. Den eneste, man bør stole på i forbindelse med download og installering af nyt software, er ophavsmanden, men naturligvis vil det også være klogest at vælge et distributionsmirror, der er forholdsvis pålideligt.

Det er normal praksis, at der ved siden af alle pakker (pakkede filer indeholdende software) ligger en tekstfil med en hashværdi af pakken. Det gør det muligt at teste pakkens integritet efter overførsel ved at beregne hashkoden på sin egen computer og derefter sammenligne. For at undgå uheldige overførselsfejl, sker der også adskillige former for tjek på dataen, mens dataen transmitteres gennem netværket. Det faktum, at dataintegritet er en af de grundlæggende byggesten til den elektroniske infrastruktur, kan der ikke herske tvivl om.



Figur 1.1: Brugerens indirekte oplevelse af dataintegritet.

1.3 Forskellige metoder til at sikre integriteten af ens data

Sikring af dataintegriteten bygger regelmæssigt på en matematisk udregning af de sendte bits/bytes. Resultat betegnes ofte som en checksum, selvom en sum normalt kun betegner resultatet af en addition. Addition er dog ikke brugt til sikring af data integritet, da det giver en alt for lille variation i checksummen, og dermed ikke er sikker nok. Dette er vist i følgende eksempel, figur 1.2, hvor checksummen af ASCII værdierne for beskeden "Hej" er udregnet.

H	e	j	checksum
71	101	106	22
(71+101+106) MOD 256 = 22			

Figur 1.2: Checksumseksempel.

Ved overførsel af en besked, har modtageren nu mulighed for at udregne om beskeden stadig giver samme checksum. I ovenstående eksempel ses, at det kun er muligt at danne checksummer fra 0 til 255, hvilket ikke er tilfredsstillende. Man kan dog ændre modulo-værdien fra 256 til et højere tal, men man vil stadig få for små variationer i checksummen, så længe addition er brugt. Af denne grund er der opfundet nogle algoritmer, til at give bedre sikkerhed af ens data's integritet. Her i blandt er de bedst kendte CRC og MD5 algoritmerne.

1.4 Troværdighed til integriteten af ens data / sikkerheden af de forskellige metoder

CRC algoritmen fungerer næsten som ved udregning en checksum ved hjælp af addition. CRC anvender istedet division, da resten af en division har langt større variation end en sum, hvilket dermed giver større mulighed for at sikre dataintegriteten. Divisoren til en CRC-checksum udregning er en slags nøgle, som vi refereré til som poly. Valget af denne værdi har stor betydning for effektiviteten af hvor vidt fejl bliver opdaget.

MD5 algoritmen foregår ud fra langt mere avancerede udregninger sammenlignet med CRC. Ofte ses MD5 i forbindelse med kryptering af kodeord eller som sikkerhedstjek i forbindelse med open-source software. Grunden til at den MD5 er ideel til kryptering af kodeord, skyldes det er en en vejs funktion, også kaldet en hashfunktion. Det er altså ikke muligt at udregne den oprindelige klartekst ud fra en cipher tekst. Til at sikre open-source software er forblevet uændret, kan man ved at hente softwaren fra en server, samt MD5 checksummen fra et andet mirrorsite, sammenligne om der er foretaget ændringer i koden.

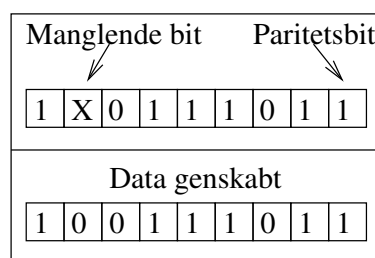
1.5 Brugen af dataintegritet til genskabelse af data

En måde man kan tjekke for dataintegritet ved overførsler, er ved hjælp af paritetsbits. Paritetsbits er "tjekbits", der er en checksum for en datastreng af en hvis længde. Oftest fungerer den ved at tælle antallet af 1'ere og 0'ere i dataen, og sætte paritetsbitten til 1, hvis der er et ulige antal 1'ere og 0, hvis den er lige. Ved brug af disse paritetsbit kan man ikke bare opdage fejl, men også genskabe de tabte data, ud fra den data man har og paritets bitten. Jo større mængde paritetsbit, jo større er muligheden for at genskabe meget tabt data. Se figur 1.3.

En metode baseret på denne teknik, Cross Interleave Reed Soloman Code (CIRC), benyttes eksempelvis på audio cd'er og dvd'er. Denne metode er istand til at genskabe så mange data, at en cd teoretisk kan afspilles fejlfrit, selv om op til 1/4 af disken fejler.

Et andet eksempel på brug af paritetsbits er i ECC (Error Correcting Code) RAM. Hver ECC er beregnet som en paritetsbit bygget op af forskellige dele af datastrengen, og ved det rigtige valg, er ECC istand til ikke alene at opdage en singlebit fejl, men også finde den rette bit og ændre den til den korrekte værdi. Den mest almindelig metode er Single Error Correction with Double Error Detection (SECDED). Denne fejlkorrigeringsmetode bygger på antagelsen at de fleste fejl (ca. 98%) kun involverer 1 bit. SECDED metoden bruger 7 bits for at fejlkorrigere en 32 bit datablok, men ved størrere datablokke, er kravet ikke tilsvarende størrere, eksempelvis kræves 8 bit for en 64 bit datablok.

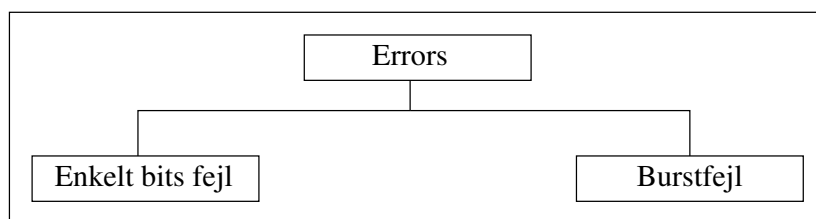
I fig 1.3 ses hvordan en paritetsbit beregnes ud fra de 8 databit. Deres binære værdi lægges sammen (her er der som udgangspunkt valgt 5 1'ere og 3 0'ere). Da antallet af 1'ere er ulige, bliver paritetsbitten 1. I dette eksempel er der tabt 1 bit, en 1'er. De 7 databits binære værdier lægges sammen, og der er nu 5 1'ere og 2 0'er. Da paritetsbitten er 1, kan der ikke være et lige antal 1'ere, og den manglende bit kan derfor findes som værende 0.



Figur 1.3: Genskabelse af tabt data.

Kapitel 2

Fejl ved transport af data

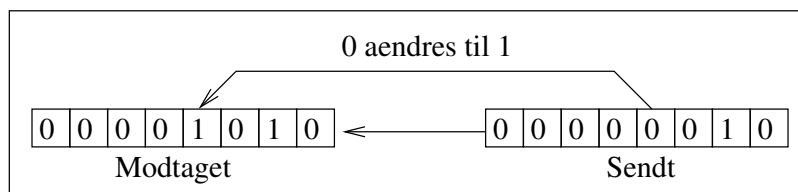


Figur 2.1: De mulige fejl der kan forekomme.

2.1 Enkelt bit fejl (Single bit error)

En enkelt bitfejl (se figur 2.1) opstår når en enkelt bit i en datablok ændres fra 0 til 1 , eller fra 1 til 0. Denne type fejl opstår ikke særligt ofte ved serielle transmissioner da en støj for det meste er i længere tid på datakanalen end datablokken.

Figur 2.2 viser en enkelt bit fejl



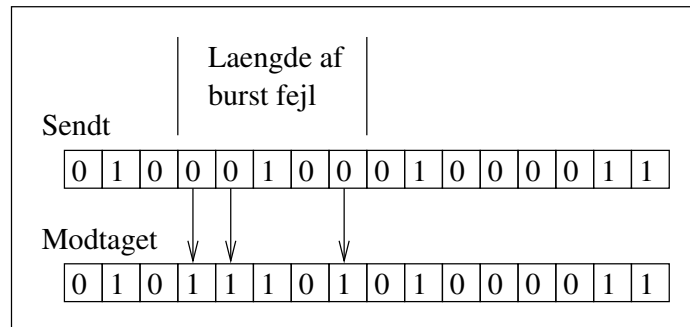
Figur 2.2: Enkeltbit fejl.

2.2 Burst fejl

En burst fejl (se figur 2.1) opstår når 2 eller flere bits i en datablok ændres fra 0 til 1, eller fra 1 til 0. En burst fejl påvirker tilfældige bits i en datablok, og opstår for det meste under serielle trans-

missioner, da støjen er længere tid på datakanalen end bitten. Antallet af fejl afhængere derfor af datamængden og støjens varighed på kanalen. Burst fejlens længde måles fra første ændret bit til den sidste , bemærk at der kan være uændret bit mellem de korupte.

Figur 2.3 viser en en burst fejl



Figur 2.3: Burstfejl.

Kapitel 3

Algoritmer

3.1 CRC (Cyclic Redundancy Check)

Formålet med udviklingen af CRC (Cyclic Redundancy Check) er at skabe en funktion, der kan lave et unikt fingeraftryk på en mængde data.

Dette fingeraftryk kan bruges til at forsikre at den rigtige datablok er blevet overført fejlfrit fra et sted til et andet, samt om det er den oprindelige data, der blev efterspurgt.

Metoden der bruges er en matematisk funktion der ud fra den aktuelle datablok genererer en unik talkombination der beskriver hvordan datablokken fuldstændt er struktureret, denne talkombination kaldes CRC-summen.

Ideen med CRC-summen er at afsenderen genererer en checksum når datablokken sendes og vedhæfter den til datablokken. Når modtageren modtager datablokken bliver den samme funktion udført på datablokken, er CRC-summen den samme som afsenderens bliver datablokken accepteret, hvis ikke bliver den slettet og den efterspørges igen.

CRC-generatoren bruger modulo-2 division som vist i tabel 3.1. Først trækkes den 4bit divisor fra de fire første bit af datablokken(dividenden), hver bit trækkes fra uden at røre den næste bit i rækken. Derefter trækkes den næste bit i rækken ned så resten bliver samme længde som divisoren, derefter fortsætter den indtil datablokken er brugt op. Skulle det ske at divisoren bliver større end resten inden man er færdig, skiftes dividenden ud med 0000, trækker den fra og hiver en ny ubrugt bit ned, og derefter fortsætter med den gamle.

CRC-tjekket fungerer på samme måde som generatoren. Når datablokken(e) modtages bliver der udført en modulo-2 division. Bliver resten af divisionen lig 0, bliver den godkendt, hvis ikke slettet og sendes igen.

```

1101 | 100100 000
      1101
      -----
        1000
        1101
        -----
          1010
          1101
          -----
            1110
            1101
            -----
              0110
              0000 <---- Nul indsættes i stedet for divisoren
              -----
                1100
                1101
                -----
                  001 <----CRC
  
```

Tabel 3.1: Eksempel på binær division i en CRC generator.

3.2 Polynomier (Polynomials)

I mange CRC-generatorer bliver divisoren repræsenteret som et aritmetisk (algebraic) polynomium. Grunden til at dette format bruges er at det er matematisk bevist, at man kan finde flere fejl i en bitstrøm, hvis man vælger det rigtige polynomium. Teorien bag de valgte polynomier der bruges, vil vi ikke komme længere ind på, dog vil vi uddrage et par få regler et polynomium skal følge for at blive effektiv, samt hvordan polynomierne repræsenterer divisoren.

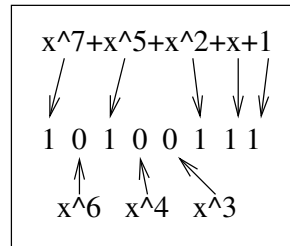
Når man vil vælge et polynomium skal det opfylde følgende kriterier:

- Må ikke være delelig med X
- Skal være delelig med $(X + 1)$

Første kriterium forsikrer, at alle burstfejl af samme grad som polynomiet bliver fundet. Det andet forsikrer, at alle fejl der påvirker et ulige tal af bits bliver fundet.

3.1 Repræsentation af en divisor, samt standard polynomier

CRC er en meget effektiv metode at finde fejl i data på, hvis de givne retningslinier for polynomiet overholdes.



Figur 3.1: Polynomisk repræsentation af divisor.

Evnen til at finde singlebitfejl samt burstfejl i mange varianter er en klar styrke og gør CRC til en af de mest brugte algoritmer til fejlfinding inden for datakommunikation og netværk. Selve implementationen af algoritmen kan gøres på mange måder, alt afhængig af den ønskede hastighed og kodekompleksitet. Dog skal man tage højde for maskinens registre, hvis man arbejder med divisorer der ikke er i form af polynomier.

3.3 MD5 (Message Digest v5)

MD5 er en såkaldt “message digest” algoritme, som bruges til at genere et digitalt fingeraftryk af noget data. Det gøres ved at dataene køres igennem en kryptografisk hashfunktion. Hashalgoritmen laver i MD5s tilfælde en 128bit værdi som repræsenterer den data, som den har regnet på. “Magien” sker vha. en del tung matematik, men ideen bag de specielt udvalgte funktioner, som man bruger til at genere hashværdien for dataene, er:

Den procedure som dataene køres igennem kalder vi H

Hvis $H(x) \neq H(y)$ så er $x \neq y$ også gældende, og hvis $H(x) = H(y)$ så er $x = y$ sandsynligvis også sandt. Problemet her ligger i at en 128bit værdi maksimum kan give 2^{128} mulige værdier, hvilket selvfølgelig er en del, men hvis man forestiller sig at der er uendeligt mange forskellige data må $H(x) = H(y)$ kunne være sandt, også selv om $x \neq y$. Dog skulle de 2^{128} mulige hashværdier og den komplekse procedure som $H(x)$ repræsenterer gøre det tilstrækkeligt svært at finde et y , der opfylder $H(x) = H(y)$ og $x \neq y$, til at det bliver anset som umuligt.

MD5 algoritmen er efter sigende sikker at bruge, selvom en gruppe forskere i 1994 fandt en måde at lave to forskellige beskeder med samme hashværdi. Den bruges bla. til at kryptere passwords, da det er umuligt at udlede beskeden ud fra hashværdien, selvom man teoretisk kan lave en tilsvarende hashværdi ved at prøve sig frem. Udover passwordkryptering bruges MD5 til at checke om filer er blevet ændret under transport, eller af en ondsindet person.

3.4 Forskellig brug mellem MD5 og CRC

Begge algoritmer bruges bla. til at checke, om data er blevet ændret under transport. CRC32 er en hurtig algoritme som fanger de fleste fejl der sker ved datatransport, men det er forholdsvist let at

ændre dataen på en sådan måde, at CRC32 algoritmen ikke opdager det. MD5 derimod bruger en kryptografiskberegningsteknik, hvilket gør den langsommere end CRC32, men det er næsten umuligt at ændre dataen på en måde, så algoritmen ikke opdager det. CRC32 algoritmen er også meget let at lave som hardware, fordi den udelukkende bruger logiske funktioner til at regne med, som findes direkte i formen af elektroniske komponenter. MD5 bruger f.eks. sinus til en del af sin udregning, som ikke findes direkte i de fleste processore. Se tabellerne 3.2 og 3.3.

MD5	
+	-
- Rimelig sikker	- Forholdsvis langsom til simpel fejlfinding - svær at implementere som hardware

Tabel 3.2: MD5.

CRC32	
+	-
- hurtig - let at implementere i hardware	- let at snyde

Tabel 3.3: CRC32.

Kapitel 4

Konklusion

Med baggrund i vores analyse mener vi, at have givet et indblik i nogle af de teorier og problemstillinger, der findes inden for emnet dataintegritet.

Ved hjælp af teorien ser vi os i stand til at kunne arbejde os hen mod et løsningsforslag, til den givne problemstilling.

Udfra analysen af CRC og MD5 algoritmerne, kan man konkludere at valget af checksumsalgoritme bør foretages udfra individuelle krav til sikkerhed og hastighed - med MD5 som den sikreste og CRC som den hurtigste.

Kapitel 5

Problemformulering

5.1 Det initierende problem

Ud fra vores problemanalyse er vi kommet frem til følgende initierende problem: "transport af data gennem usikre kanaler".

5.2 Problemafgrænsing

Fordi "transport af data gennem usikre kanaler" er et utroligt stort emne, har vi valgt, på grundlag af interesse og tid, at afgrænse det til sikker transport af filer. Vi har ydermere afgrænset emnet ned til følgende spørgsmål, der kan danne baagrund for et opfølgende projekt:

- Hvordan kan datatransmissioner sikres?
- Hvordan finder man fejl i datatransmissioner?
- Hvordan kan man autokorrigere fejl i datatransmissioner?
- Hvordan kan man sikre integriteten af dataen?

5.3 Problemformulering

På baggrund af følgende scenarie:

Et firma ønsker at overføre store filer mellem 2 afdelinger på en sikker og hurtig måde. Firmaet benytter sig af en krypteret dataforbindelse, som koster for meget i trafik, og ønsker derfor en anden løsning.

Vi vil arbejde os frem til et eller flere løsningsforslag, som firmaet kan tage i brug. Disse forslag vil vi hovedsagligt bygge ud fra eksisterende teknologier, som ssl og md5.

Kapitel 6

Kildeliste

6.1 Bøger

Data communications and Networking 2nd edition
Forfatter: Behrouz A. Forouzan
Udgivelsesår: 2001
ISBN: 0-07-118160-1

6.2 Internetadresser

Titel: Understanding Cyclic Redundancy Check
Dato: 16/9-03
URL: <http://www.4d.com/docs/CMU/CMU79909.HTM>

Titel: Wikipedia (the free encyclopedia) - Checksum
Dato: 16/9-03
URL: <http://www.wikipedia.org/wiki/Checksum>

Titel: Wikipedia (the free encyclopedia) - Hash function
Dato: 16/9-03
URL: http://www.wikipedia.org/wiki/Hash_function

Titel: Wikipedia (the free encyclopedia) - MD5
Dato: 16/9-03
URL: <http://www.wikipedia.org/wiki/MD5>

Titel: Checksum
Dato: 16/9-03
URL: <http://www.math.utah.edu/~beebe/software/filehdr/node15.html>

Titel: CRC Explained - a painless guide to CRC error detection algorithms

Dato: 16/9-03

URL: <http://w3.mech.uwa.edu.au/~petitj01/FTS/CRC%20Explained.htm>

Titel: Wikipedia (the free encyclopedia) - Message digest

Dato: 16/9-03

URL: http://www.wikipedia.org/wiki/Message_digest

Titel: RFC1321 - The MD5 Message-Digest Algorithm

Dato: 16/9-03

URL: <http://www.ietf.org/rfc/rfc1321.txt>

Titel: Error Correction

Dato: 16/9-03

URL: <http://csunix1.lvc.edu/~snyder/correct.html>