

Integritet af data og elektromagnetisk stråling

Studieområdeprojekt (SOP)

Carl Benjamin S. Dreyer

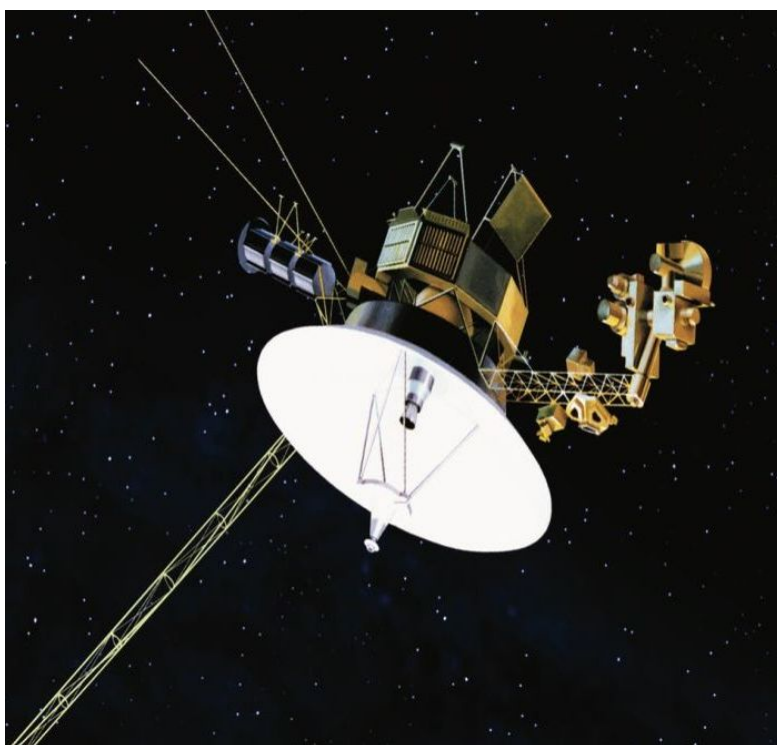


Figure 1: Voyager 2

Udarbejdet på HTX i forbindelse med Studieområdeprojektet (SOP) i 3G

Vejledere: Kasper Zøllner & Daniel Peter Withenstein

Skole: NEXT Københavns Mediegymnasium (KMG)

Fag: Fysik A & Programmering B

Afleveret: d. 19 dec. 2023

Anslag: ca. 39.000 (16ns)

Integritet af data og elektromagnetisk stråling

Carl Benjamin S. Dreyer

Resumé

Dette studieområdeprojekt beskæftiger sig med integriteten af data og elektromagnetisk stråling - specielt hvordan man kan bevare integriteten. Der undersøges hvordan fysisk beskyttelse - *radiation hardening*, og software i form af fejl korrigeringskoder, kan forholde integriteten af data og formindske data korruption. Den fysiske analyse består af at teste forskellige materials halveringstykkelser og hvordan man kan forudsige hvornår et radioaktivt stof henfald. Der blev blandt andet fundet ud af at dense materialer som bly beskytter godt mod stråling og at man statistisk kan forudsige hvornår et radioaktivt stof henfalder. Derudover bliver der analyseret og implementeret to fejl korrigeringsalgoritmer, som efterfølgende sammenlignes og diskuteres. Baseret på analysen, viser det sig at Reed-Solomon fejl korrigeringskoden er mere kraftfuld end hamming koden. I diskussionen viser det sig at både fysisk beskyttelse i form af *radiation hardening*, og software beskyttelse i form af fejl korrigeringskoder, hver har deres formål og løser problemet forskelligt. Det blev tydeliggjort at den bedste kombination af fysisk- og software beskyttelse kraftigt afhænger af situationen og ens formål.

Problemformulering

-Som beskrevet på netproever.dk

Hvordan kan man beskytte data - lagret eller i transmission - mod elektromagnetisk stråling?

- Redegør for hvorfor elektromagnetisk stråling er et problem overfor integriteten af data. Herunder kom ind på digital kommunikation, partikelfysikken bag stråling og Error Correcting Codes (ECCs).
- Analysér forskellige beskyttelsesmetoder mod data korruption
 - Simple metoder: Forbedre Signal to Noise Ratio (SNR), Checksums, Automatic Repeat Requests (ARQs), repetition codes.
 - ECCs: hamming codes, reed solomon codes.
 - Fysisk beskyttelse: Undersøg strålings påvirkning på lagret data, og undersøg forskellige materials beskyttelsesevne, i form af deres halveringstid.
- Analysér og implementér to forskellige ECC algoritmer.
- Diskutér effekten af de forskellige beskyttelsesmetoder (fysisk- og software beskyttelse) der er blevet undersøgt, samt sammenlign og diskutér to forskellige ECC algoritmer.

Indholdsfortegnelse

Resumé	i
Indledning	1
Radioaktivitet & Stråling	2
Radioaktiv henfald	2
α -henfald	3
β -henfald	3
γ -henfald	3
Aktivitet & henfaldsloven	4
Absorption & halveringstykkelser	5
Fejl korregeringskoder	6
Detektion	7
Lineære koder	7
Generator Matrix	9
Parity-Check Matrix	9
Cykliske koder	11
Analyse af fysikken	12
Materialers halveringstykkelser	12
Forudsigelse af ustabile kernerers henfald	15
Barium-137 halveringstid	17
Analyse af fejl korregeringskoder	18
Repetition Koder	18
Hamming Koder	20
Reed-Solomon kode	21
Design & Implementation	22
Hamming codec	23
Enkodning	24
Dekodning	26
Reed-Solomon codec	28

Diskussion	30
Konklusion	31
Kildeliste	32
Bilag	36
Bilag 1 - Sandsynlighed for tælleletal mellem 40 og 60	36
Bilag 2 - Udledning af minimum hamming distance	36
Bilag 3 - Mængde af fejl (7,4) Hamming kode kan detektere og korrigere	37
Bilag 4 - Opslagstabel for fejl korrigering	38
Bilag 5 - Energineavu af Cæsium henfald	39
Bilag 6 - Generering af alle kodeord ud fra generator polynomie	39
Bilag 7 - Detaljeret process af erasure korrigering	40
Bilag 8 - Kildekode af hamming (7,4) codec	40
Bilag 9 - Kildekode af RS codec	41
Bilag 10 - Test af hamming koden	41
Bilag 11 - Bit flip program	42

Indledning

Stråling kan være en trussel mod elektroniske kredsløb idet at det kan slå elektroner løse¹. Richard Hamming (1915-1998), opfinderen af den populære hamming fejl korrigeringskode, sagde at det for et digitalt system kan betyde kritiske fejl og lede til et komplet invalidering af systemet². Dette er et problem som påvirker alt fra rumfartøjer³ og kvantecomputere⁴, til ens personlige computer⁵. På grund af det, er der gennem årene blevet udviklet forskellige beskyttelsesmetoder til at bevare integriteten af data, i form af *radiation hardening* - fysisk beskyttelse, og fejl detektions- og korrigeringskoder. Det er derfor interessant at undersøge hvordan de forskellige metoder, som forsøger at løse det samme problem, virker og hvordan de hver løser problemet på vidt forskellige måder.

Opgaven starter med at redegøre for den baggrundsviden som er vigtig for opgavens analyse. Den starter med at redegøre for kernefysikken og radioaktiv henfald og går derefter hen til at redegøre for fejl detektions- og korrigeringskoder. Redegørelsen sætter fundamentet for en analyse af materialers beskyttelsesevne, og naturens tilfældige henfald. Derudover igangsættes en analyse af repetition, hamming og Reed-Solomon fejl korrigeringskoder, som indleder til en reel implementation af hamming (7,4) koden og Reed-Solomon koden. Til sidst diskuteres de forskellige beskyttelsesmetoder, samt de to implementationer af fejl korrigeringskoder, og der konkluderes på projektet som helhed.

¹Yu et al. 2010, s. 1.

²Hamming 1950, s. 147.

³Poivey 2019, s. 2.

⁴Vepsäläinen et al. 2020.

⁵Harrell 2010.

Radioaktivitet & Stråling

Atomkernen er opbygget af protoner og neutroner, som henholdsvis består af 2 up-kvarer, 1 downkvar og 2 downkvarer, 1 upkvar⁶. I Niels Bohrs (1885-1962) atom-model sværmer elektronerne i en række endelige skaller omkring kernen. Hvert grundstof har et unikt antal af protoner, Z - atomnummeret, hvor neutrontallet N , bestemmer isotopen. Ofte angiver man atomkernen for et grundstof X , som:

$${}^Z_A X \quad (1)$$

Hvor $A = Z + N$ er nukleontallet - antallet af neutroner og protoner⁷. En proton har ladningen $+e$ og en elektron har en ladning på $-e$, hvor e er elementarladningen. Neutroner har ingen ladning. Coulombs Lov beskriver den elektrostatiske kraft F_E , mellem to punkt ladede partikler q og Q ⁸:

$$F_E = \frac{1}{4\pi\epsilon_0} \frac{q \cdot Q}{r^2} \quad (2)$$

Hvor ϵ_0 er vakuumpermittiviteten, q og Q er ladningen for de to partikler og r er afstanden mellem partiklerne. Ud fra Coulombs lov (2) skulle man tro at protonerne i atomkernen ikke ville kunne blive så tæt sammen, men de holdes sammen af en stærkere kraft kaldt den stærke kernekraft⁹. Den stærke kernekraft virker mellem nukleoner - protoner og neutroner, og har en meget lille rækkevidde.

Radioaktiv henfald

Ved små atomkerner dominerer den stærke kernekraft og holder atomkernen stabil. Men ved større atomkerner med flere protoner kan Coulombs lov træde i kraft og gøre atomkernen ustabil - den bliver radioaktiv. Det er derfor at der i større atomkerner ofte skal være flere neutroner end protoner, fordi de bidrager til den stærke kernekraft, men ikke til den elektrostatiske kraft. Når en atomkerne er radioaktiv - ustabil, kan den henfalde ved udledning af stråling. Strålingen kan bestå af partikler eller elektromagnetisk stråling (EM-bølger). Strålingen for radioaktive kerner kan opdeles i 3 typer; α -stråling,

⁶Michelsen and Pedersen 2019, kap. 10.1.2.

⁷Holc, Lund, and Kraaer 2021, kap 7.1.

⁸Holc, Lund, and Kraaer 2021, kap. 3.1.

⁹Holc, Lund, and Kraaer 2021, kap 7.1.

β -stråling og γ -stråling. De 3 typer stråling er typisk meget energirige og er i stand til at slå elektroner ud af atomer¹⁰ og potentielt interferere med data lagret eller i transmission¹¹. α - og β -stråling består af partikler, mens γ -stråling består af fotoner, som kan anses som at være det samme som EM-bølger pga. partikel-bølge dualiteten¹².

α -henfald

Ved α -henfald skilder moderkernen sig af med to protoner og 2 neutroner - en heliumkerne ${}^4_2\text{He}^{2+}$. Eksempelvis kan Radium-226 henfalde til Radon-222, ved udsendelse af en α -partikel:



β -henfald

I β -henfald laves en neutron om til en proton, og der udsendes en elektron og en antineutrino $\bar{\nu}_e$. Cæsium-137 er et eksempel på en radioaktiv kerne, med sandsynlighed for at henfalde til Barium-137:



En neutron i Cæsium-137 bliver omdannet til en proton, hvilket gør atomnummeret en gang større.

γ -henfald

I eksemplet foroven (4) henfalder Cæsium-137 til Barium-137. ${}^{137}_{56}\text{Ba}^*$ er i exciteret tilstand, hvilket betyder at kernen har for meget energi til at være stabil¹³ (Se evt. **Bilag 5 - Energineavu af Cæsium henfald**). Derfor henfalder den ved udsendelse af γ -stråling:



¹⁰Holc, Lund, and Kraaer 2021, kap 7.2.

¹¹Overill 2020, s. 1.

¹²Feynman 2007, vol 3, kap 2.

¹³Holc, Lund, and Kraaer 2021, kap 7.2.

Der forekommer altså ingen grundstofforvandling.

Aktivitet & henfaldsloven

Aktiviteten A , for en radioaktiv kerne beskriver antallet af henfald pr. sekund. På grund af henfaldets tilfældige natur, er det umuligt at vide præcist hvornår en radioaktiv kerne vil henfalde¹⁴. Dog kan man udtale sig om sandsynligheden for at kernen henfalder indenfor et tidsrum¹⁵. Given et radioaktivt stof med antallet af atomkerner N , så er aktiviteten defineret som:

$$A = -\frac{dN}{dt} \quad (6)$$

Hvor $-dN$ svarer til ændringen af atomkerner i stoffet - antallet af kerner der henfalder i tidsrummet dt . Aktiviteten måles i $\text{Bq} = \text{s}^{-1}$. Når et henfald sker, falder antallet af kerner i det radioaktive stof, hvilket statistisk set hentyder til at aktiviteten A , er ligefrem proportional med tiden t ¹⁶. Lad os betragte et radioaktivt stof, som ved tidspunktet $t = 0\text{s}$ består af N_0 antal kerner. I et kort tidsrum dt , vil det gælde at ændringen i antallet af kerner dN , vil være proportional med mængden af kerner der er N , og dt :

$$dN = -k \cdot N dt \quad (7)$$

Dette er en differentialligning, som beskriver ændringen i antallet af kerner i det radioaktive stof, hvor k er proportionalitetskonstanten, kaldt henfaldskonstanten, som afgør hvor hurtigt stoffet henfalder. Ved løsning af differentialligningen kan antallet af kerner N , beskrives som en funktion af tiden t . Det er en differentialligning på formen $y' = ay$,

¹⁴Holc, Lund, and Kraaer 2021, kap 7.3.

¹⁵Michelsen and Pedersen 2019, kap 10.4.1.

¹⁶Øhlenschläger 2018, kap 10.5.

hvis løsning er en eksponentiel funktion¹⁷:

$$\begin{aligned}
 \int_{N_0}^N \frac{dN}{N} &= - \int_0^t k dt \\
 [\ln(N)]_{N_0}^N &= - [k \cdot t]_0^t \\
 \ln(N) - \ln(N_0) &= -k \cdot t \\
 \ln\left(\frac{N}{N_0}\right) &= -k \cdot t \\
 N &= N_0 \cdot e^{-k \cdot t}
 \end{aligned} \tag{8}$$

Dette kaldes for henfaldsloven, og beskriver antallet af moderkerner N , i det radioaktive stof, som falder eksponentielt med tiden t . Halveringskonstanten for denne funktion er halveringstiden $T_{\frac{1}{2}}$, som beskriver hvor langt tid der går før N er halveret¹⁸:

$$\begin{aligned}
 N(T_{\frac{1}{2}}) &= \frac{1}{2} \cdot N_0 \\
 \frac{1}{2} \cdot N_0 &= N_0 \cdot e^{-k \cdot T_{\frac{1}{2}}} \\
 \frac{1}{2} &= e^{-k \cdot T_{\frac{1}{2}}} \\
 \ln\left(\frac{1}{2}\right) &= \ln\left(e^{-k \cdot T_{\frac{1}{2}}}\right) \\
 \ln(1) - \ln(2) &= -k \cdot T_{\frac{1}{2}} \\
 T_{\frac{1}{2}} &= \frac{\ln(2)}{k}
 \end{aligned} \tag{9}$$

Halveringstiden er altså omvendt proportional med henfaldskonstanten k .

Absorption & halveringstykkelser

Når strålingen interagerer med materialer bruger man begrebet intensitet I , som er målt i $\frac{W}{m^2}$, og beskriver strålingens effekt pr. areal når strålingen passerer gennem en flade. α - og β -stråling har en relativ lille gennemtrængningsevne, mens γ -stråling har en betydelig større. Ved passering igennem et materiale, svækkes γ -strålingen afhængig

¹⁷Jensen, Marthinus, and Hansen 2023, kap. 4.3.1.

¹⁸Michelsen and Pedersen 2019, kap 10.4.2.

af fotonenergien og materialet¹⁹. Det viser sig at intensiteten $I(x)$ aftager eksponentielt med tykkelsen af materialet x ligesom i henfaldsloven (8):

$$I(x) = I_0 \cdot e^{-\mu \cdot x} \quad (10)$$

Hvor I_0 er intensiteten ved $x = 0m$ og μ er absorptionskoefficienten. Absorptionskoefficienten μ , afhænger af fotonens energi og materialet, og kan findes ved opslag i en databog. Ud fra udledningen af halveringstiden (9), kan man se at halveringstykkelsen kan skrives som:

$$x_{\frac{1}{2}} = \frac{\ln(2)}{\mu} \quad (11)$$

Fejl korrigeringskoder

Error Correcting Codes (ECC) eller fejl korrigeringskoder er en metode man kan bruge i software eller hardware til at beskytte integriteten af data, som sørger for at korrigere fejl ved eventuelle bit-flip, som kan forekomme under transmission²⁰. Teorien bag fejl korrigeringskoder kan repræsenteres på mange forskellige måder; både geometrisk og algebraisk. Undersøgelsen af fejl korrigeringskoder kræver mange matematiske teorier indenfor abstrakt algebra, der vil bl.a. blive anvendt mængdelære (sæt teori), gruppe teori, endelige legemer (finite fields), vektorrum og matricer til at beskrive koderne.

Når der snakkes om fejl korrigeringskoder indenfor computer systemer arbejdes der med tal i det endelige legeme \mathbb{F}_2 - altså 0 eller 1, fordi computere kun kan forstå binær. Matematiske operationer udføres i modulu 2, hvilket deler samme karakteristika som bitwise operatøren xor (\oplus). Eksempelvis er $1 + 1 \pmod{2} = 1 \oplus 1 = 0$. \mathbb{F}_2^n er et n -dimensionelt vektorrum over \mathbb{F}_2 , hvor hvert element i \mathbb{F}_2^n kan beskrives som en n -tuple (a_1, a_2, \dots, a_n) , hvor $a_i \in \mathbb{F}_2$. Eksempelvis hvis $n = 3$, så vil \mathbb{F}_2^3 indeholde alle mulige 3-tuple fra \mathbb{F}_2 såsom $(0, 0, 0)$, $(1, 0, 1)$ og $(0, 0, 1)$. Disse vektorer kan repræsentere bitsekvenser, f.eks. kan binær beskeden 010 blive repræsenteret som en 3-tuple $(0, 1, 0)$ i \mathbb{F}_2^3 vektorrummet. I de kommende afsnit vil det vise sig at det er nyttigt at anvende disse legemer og vektorrum til at beskrive fejl korrigeringskoder.

¹⁹Michelsen and Pedersen 2019, kap 10.4.3.

²⁰Mitchell 2009, s. 1.

Detektion

Der er stor forskel fra at kunne detektere en fejl, til at kunne rette den. Fejl detektion er når der kan påvises at der er sket en fejl - et bit-flip er forekommet. Det betyder nødvendigvis ikke at man kan rette fejlen. Selvom man ikke kan rette fejlen, er det stadig nyttigt information, fordi det giver modtageren mulighed for at anmode om en gen-transmission²¹.

Fejl detektionskoder, kan variere meget i kompleksitet, afhængig af hvor mange fejl den skal kunne detektere, og hvor effektiv den skal være - hvor meget ekstra redundans der skal tilføjes. Den simpleste form for fejl detektion er at tilføje 1 ekstra bit, kaldt en paritet/redundans bit, til slutningen af ens data. Denne metode er kaldt CRC-1 og er en af de simpleste algoritmer i samlingen af Cyclic Redundancy Checks (CRCs)²². Given en streng af data af længden k , vil den totale længde af dataet sendt have længden $k + 1$. Paritet bitten beskriver om summen af alle bitsne i dataet - den såkaldt hamming weight, som forklares senere - er lige eller ulige²³ ²⁴. Når en modtager får en besked, beregner den summen af beskeden, og sammenligner med paritet bitten, og hvis de ikke passer, så er der sket en fejl. Hvis modtageren detekterer at der er sket en fejl, anmoder den tit om en gen-transmission, som f.eks. Hybrid Automatic Repeat reQuest (HARQ), som bliver anvendt i mange trådløse protokoller²⁵. Fejl detektion består altså af at beregne en cheksum og tilføje den til dataet man sender over kanalen. Modtageren laver samme beregning, og sammenligner checksummene. Hvis der er fejl, kræver det at der er en feedback kanal - en kanal hvor modtageren kan sende en besked tilbage til afsenderen, for at bede om en gen-transmission.

Lineære koder

Lineære koder strækker sig over en række binære koder som bliver brugt i Forward Error Correction (FEC). Hvis der skulle forekomme et bit-flip under transmission, så har modtageren nok information, til at kunne rette fejlen selv uden af at bede senderen om en gen-transmission²⁶. Lineære koder virker ved at sende flere bits end hvad dataet

²¹Ryan and Lin 2009, s. 3.

²²Cook 2023.

²³Ziemer and Tranter 2006, s. 638-639.

²⁴Glover and Dudley 1991, s. 1.

²⁵Marks 2003, s. 5.

²⁶Wang, Sklar, and Johnson 2002.

består af - der bliver tilføjet ekstra redundans/paritet bits, som bruges til at kunne detektere og rette fejl²⁷.

Dermed ved enkodning af en sekvens af bits t , kaldt for en kode, med længden k , vil den genererede kodeord c , have en længde n , hvor $n > k$. Kodeordet c , er hvad der vil blive sendt over kanalen til modtageren. For enhver lineær kode kan man beregne den såkaldt hamming vægt og hamming distance. Hamming vægten af en bitstreng er antallet af aktiveret bits:

$$wt(t) = \sum_{i=0}^{n-1} wt(t_i), \quad \text{hvor } wt(t_i) = \begin{cases} 0, & x_i = 0 \\ 1, & x_i \neq 0 \end{cases} \quad (12)$$

Hvis $t = 1010$, så vil hamming vægten være $wt(t) = 2$, fordi der er to 1'ere i t . Given en (7,4) lineær kode $C \subset \mathbb{F}_2^n$, hvor tallene i paranteserne referer til mængden af data bits $k = 4$, og den totale længde efter enkodning $n = 7$. Ved enkodning af alle mulige k kombinationer af bits $\forall t \in \mathbb{F}_2^k$, vil de genererede kodeord $\forall c \in C$, blive:

t	c	t	c	t	c	t	c
0000	0000000	0100	0100110	1000	1000101	1100	1100011
0001	0001011	0101	0101101	1001	1001110	1101	1101000
0010	0010111	0110	0110001	1010	1010010	1110	1110100
0011	0011100	0111	0111010	1011	1011001	1111	1111111

Figure 2: Tabel over alle kodeord i (7,4) Hamming koden

Hvert kodeord adskiller sig med 3 bits fra hinanden, hvilket kaldes for hamming distancen²⁸. Hamming distancen, d , siger noget om hvor mange fejl en kode kan detektere og korrigere²⁹:

$$e_d = d - 1 \quad (13)$$

$$e_k = \left\lfloor \frac{d-1}{2} \right\rfloor \quad (14)$$

Hvor e_d denoterer hvor mange fejl koden kan detektere og e_k denotere mængden af

²⁷Mackay 2003, s. 9.

²⁸Hamming 1950, s. 154-155.

²⁹Danziger 2005, s. 2.

fejl koden kan korrigere (Se evt. **Bilag 3 - Mængde af fejl (7,4) Hamming kode kan detektere og korrigere**).

Lad C være en (n, k) lineær kode, hvor $C \subset \mathbb{F}_2^n$. En lineær kombination mellem enhver kodeord c_1, c_2, \dots, c_n i C , vil resultere i et kodeord $x \in C$. Med andre ord, så er summen af enhver par af kodeord endnu et tilladt kodeord^{30 31}.

Generator Matrix

En generator matrice er en matrice brugt til at generere et kodeord ud fra en bitsekvens - den er ansvarlig for at encode dataet. Given en lineær kode på formen (n, k) vil generator matricen se ud på formen³²:

$$G = [I|S] \quad (15)$$

Hvor G er generator matricen, I er identitets matricen med k kolonner og k rækker, og S er den sidste del, som er k høj og $n - k$ bred. Given en lineær kode t så vil den generede kodeord, c blive:

$$c = t \cdot G \quad (16)$$

For eksempel given en $(n = 7, k = 4)$ lineær kode - der er 4 information bits, vil $t = [b_0 \ b_1 \ b_2 \ b_3]$. Generering af et kodeord er altså en matrice multiplikation mellem information bitsne, og den lineære kodes generator matrice. Efter koden er blevet enkodet, er den klar til at blive sendt til en modtager.

Parity-Check Matrix

Nu, når vi ved hvordan en kode bliver enkodet til et kodeord (16), hvordan dekoder man det til den oprindelige kode? Og hvad hvis der er sket et bit-flip, hvordan rettes fejlen? Til dette bruges parity-check matricen, H , som står for at udpege om der er sket en fejl, og hvis ja, hvor fejlen fandt sted. Det gælder for en hvilken som helst kodeord $c \in C$ at

³⁰Sena 2021, s. 8.

³¹Ryan and Lin 2009, s. 4.

³²Goemans 2015, s. 3.

matrice multiplikation med parity-check matricen er nulvektoren³³ ³⁴:

$$H \cdot c^T = 0, \quad c \in C \quad (17)$$

Hvor $C \subset \mathbb{F}_2^n$, og c^T er kodeordet på transponeret form. Hvis $c \notin C$, bliver produktet af matrice multiplikation ligmed syndrom vektoren:

$$H \cdot c^T = s, \quad c \notin C \quad (18)$$

$$s \neq 0 \quad (19)$$

Syndrom vektoren fortæller hvilken bit i c der er fejl i³⁵. Given en (n, k) lineær kode $C \subset \mathbb{F}_2^n$, og et kodeord $c \in C$, som bliver sendt over en datakanal, hvor en fejl $e \in \mathbb{F}_2^n$, hvor $wt(e) > 0$ sker. Så vil modtageren modtage kodeordet $z = c + e \pmod{2}$. Modtageren vil tjekke det modtaget kodeord imod H , og konkludere at der sket en bit fejl, da:

$$H \cdot z^T = s, \quad s \neq 0$$

Modtageren vil kunne dekode den oprindelige besked, c , selvom der er sket en fejl, e . Det kan modtageren bl.a. gøre ved at slå syndromet op i en opslagstabel, hvor syndromer korresponderer til en fejl - i dette tilfælde ville s korresponderer til e ³⁶, hvilket betyder at fejlen kan korrigeres: $z + e = c \pmod{2}$.

H kan genereres ud fra G , da følgende gælder (15):

$$G = [I|S]$$

$$H = [S^T|I] \quad (20)$$

For en (n, k) lineær kode, vil den have generator matricen med dimensionerne $k \times n$, og parity-check matricen vil have dimensionerne $n - k \times k$.

³³Ryan and Lin 2009, s. 96.

³⁴Huffman and Pless 2003, s. 4.

³⁵Beutelspacher and Rosenbaum 1998, s. 188.

³⁶Beutelspacher and Rosenbaum 1998, s. 190.

Cykliske koder

I lineære koder er kodeord blevet matematisk beskrevet som vektorer, som befinder sig i vektorrummet \mathbb{F}_2^n . I cykliske koder bliver kodeord beskrevet som polynomier. Given et kodeord på formen b_0, b_1, \dots, b_n , vil polynomiet være: $b_0x^0 + b_1x^1 + \dots + b_nx^n$. Hvert bit i kodeordet korresponderer altså til polynomiets koefficienter. Polynomiet er på standard form, når alle x ledene går fra laveste orden (venstre) mod højeste orden (højre). Som tidligere beskrevet, så står generator matricen i (n, k) lineære koder for at enkode en binær kode $t \in \mathbb{F}_2^k$ til et kodeord $c \in \mathbb{F}_2^n$. Generator matricen *løfter* dimensionen af koden fra k , til n ved enkodning. F.eks. med $(7, 4)$ hamming koden løfter generator matricen koden fra et 4D vektorrum \mathbb{F}_2^4 til et 7D vektorrum \mathbb{F}_2^7 . Ligesom normale tal, kan man gange, dividere og bruge modulær aritmetik på polynomier. Forskellige polynomier kan klassificeres i forskellige sæt. F.eks. indeholder $\mathbb{R}[x]$ alle polynomier hvor koefficienterne befinder sig i \mathbb{R} , og hvor den ubekendte er x . Da der arbejdes med binær, vil de fleste polynomier tilhøre $\mathbb{Z}_2[x]$ el. $\mathbb{Z}[x]/2\mathbb{Z}$, som begge repræsenterer alle heltal modulo 2, altså 0 og 1. For modulær aritmetik med polynomier bliver det skrevet som $\mathbb{Z}_2[x]/(x^5 - 1)$, hvor \mathbb{Z}_2 beskriver koefficienterne og det inde i parantesen beskriver hvad der ækvivalent med 0. Fra dette eksempel kan man se at $x^5 - 1 \equiv 0$, hvilket betyder $x^5 \equiv 1$. For cykliske koder, gælder det at en cyklisk skift i kodeordet er endnu et tilladt kodeord³⁷. F.eks. kodeordet 10011 cyklisk skiftet mod højre 1 gang er 11001 - alle bits er skiftet 1 gang mod højre, hvor bitsne ved slut vikler sig rundt til starten. Når et polynomie ganges med x svarer det til et cyklisk skift mod højre. Antag kodeordet $c = b_0b_1\dots b_{n-1}$. På polynomie form vil det være: $c(x) = b_0 + b_1x + \dots + b_{n-1}x^{n-1} \in \mathbb{Z}_2[x]/(x^n - 1)$. Kodeordet ganges nu med x : $x \cdot c(x) = b_0x + b_1x^2 + \dots + b_{n-1}x^n$. Da $x^n \equiv 1$ kan det ses at polynomiet kan reduceres til: $x \cdot c(x) = b_{n-1} + b_0x + b_1x^2 + \dots + b_{n-2}x^{n-1}$, hvilket på binær form er $b_{n-1}b_0b_1\dots b_{n-2}$, som er et cyklisk skift mod højre. To cykliske skift svarer altså til $x^2 \cdot c(x)$ og 3 svarer til $x^3 \cdot c(x)$ osv. En cyklisk kode, kan også være lineær idet at en lineær kombination mellem kodeord er endnu et tilladt kodeord³⁸,

³⁷Eigenchris 2019, ep. 3b.

³⁸Eigenchris 2019, ep. 3b.

hvilket betyder for lineære cykliske koder:

$$\begin{aligned} c(x) + xc(x) + x^2c(x) + \dots \\ = (1 + x + x^2 + \dots)c(x) \\ = u(x)c(x) \end{aligned} \tag{21}$$

En polynomie $u(x)$, ganget med et kodeord $c(x)$, er altså endnu et tilladt kodeord. Generator polynomiet er et polynomie af en cyklisk kode af mindste orden $\min(\deg(c))$, hvor $c \neq 0$ ³⁹. Generator polynomiet kan bruges til at generere et kodeord fra en besked:

$$c(x) = \frac{t(x)}{g(x)} \tag{22}$$

Hvor $t(x)$ er beskeden der skal enkodes og $g(x)$ er generator polynomiet. Divisionen sker over det endelige legeme.

Analyse af fysikken

Der er større chance for data korruption når dataet bliver udsat for mange højenergi bølger. Derfor er det oplagt at undersøge forskellige materials beskyttelsesevne og hvordan man kan forudsige hvornår en radioaktiv kilde vil henfalde.

Materials halveringstykkel

Som tidligere beskrevet, så svækkes en γ -stråle når den passerer gennem et materiale, som er afhængig af bølgens energi og materialet. Materialets indflydelse kan dermed undersøges hvis bølgens energi holdes konstant. I eksperimentet bliver der anvendt et Geiger-Müller rør (GM-rør), som kan måle antallet af henfald. Der tages udgangspunkt i reaktionerne beskrevet tidligere hvor cæsium-137 henfalder til barium-137 i exciteret tilstand (4), som udsender γ -stråling, ved at energiniveauet falder til en ikke-exciteret tilstand (5). I forsøgsopstilling sættes cæsium-137 kilden til at pege hen mod et GM-rør i en fast afstand, som er tilsluttet til en LabQuest, der indsamler antallet af henfald GM-røret detekterer - kaldt for tællertallet T . Mellem GM-røret og den radioaktive kilde, indsættes plader af det materiale som undersøges i faste intervaller. Efter indsamlingen af

³⁹Pavert 2011, s. 24.

tælleallene for de forskellige tykkelser af forskellige materialer, måles baggrundstrålingen og tallene korrigeres efter baggrundstrålingen. Da antallet af moderkerne N hele tiden falder, er det vigtigt at gamma-kildens halveringstid er meget høj, så sandsynligheden for at den henfalder (stort set) er den samme gennem hele eksperimentet. Cæsium-137 har dog en halveringstid på 30,2 år hvilket betyder at den har minimal indflydelse⁴⁰. Idet tælleallene er ligefrem proportional med intensiteten som følger en eksponentiel vækst (10), fås:

$$T(x) = T_0 \cdot e^{-\mu \cdot x} \quad (23)$$

Og da tælleallene T , følger en eksponentiel vækst med tykkelsen x , er det oplagt at lave en eksponentiel kurvetilpasning til dataet:

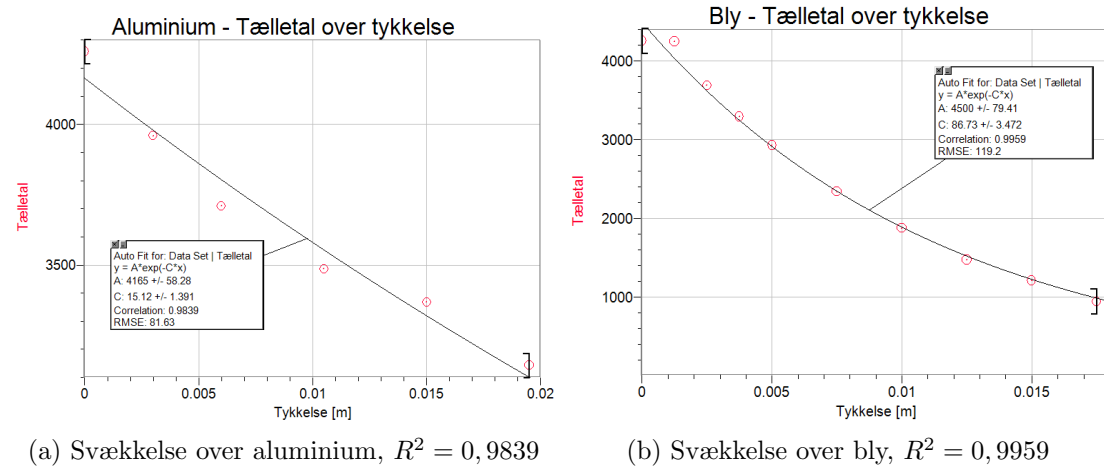


Figure 3: Tælleallene falder eksponentielt afhængig af tykkelsen

Tælleallene blev målt i 2 minutter 3 gange for hvert tykkelse og den gennemsnitlige værdi blev så taget. Så grafen viser den gennemsnitlige antal henfald GM-røret påviste indenfor 2 minutter i forhold til tykkelsen. Ud fra kurvetilpasningerne fra Logger Pro⁴¹,

⁴⁰Delacroix, Guerre, and Leblanc 2002, s. 114.

⁴¹Vernier 2021.

fås:

$$T_{\text{aluminium}}(x) = 4165 \cdot e^{-15,12x} \quad (24)$$

$$T_{\text{bly}}(x) = 4500 \cdot e^{-86,73x} \quad (25)$$

Udfra funktionerne (24) og (25), kan den observerede absorptionskoefficient aflæses $\mu_{\text{aluminium}} = 15,12\text{m}^{-1}$ og $\mu_{\text{bly}} = 86,73\text{m}^{-1}$, og halveringstykkelserne kan beregnes med (11):

$$\text{Al}_{x_{\frac{1}{2}}} = \frac{\ln(2)}{15,12\text{m}^{-1}} \approx 45,84\text{mm} \quad (26)$$

$$\text{Pb}_{x_{\frac{1}{2}}} = \frac{\ln(2)}{86,73\text{m}^{-1}} \approx 7,992\text{mm} \quad (27)$$

Den teoretiske absorptionskoefficient kan som sagt findes ved opslag i en databog. Når den exciteret barium-137 henfalder, har største delen af γ -strålingen en energi på 662keV ⁴². Baseret på NISTs XCOM database er masseabsorptionskoefficienten $\frac{\mu}{\rho}$, for bly ved 662keV , $\frac{\mu}{\rho} = 0,01035\frac{\text{m}^2}{\text{kg}}$ og for aluminium $\frac{\mu}{\rho} = 0,007433\frac{\text{m}^2}{\text{kg}}$ ⁴³, som kan konverteres til absorptionskoefficient ved at gange med materialets densitet $\rho_{\text{bly}} = 11350\frac{\text{kg}}{\text{m}^3}$ og $\rho_{\text{aluminium}} = 2699\frac{\text{kg}}{\text{m}^3}$ ⁴⁴.

$$\mu_{\text{aluminium}} = 0,007433\frac{\text{m}^2}{\text{kg}} \cdot 2699\frac{\text{kg}}{\text{m}^3} = 20,061667\text{m}^{-1} \quad (28)$$

$$\mu_{\text{bly}} = 0,01035\frac{\text{m}^2}{\text{kg}} \cdot 11350\frac{\text{kg}}{\text{m}^3} = 117,4725\text{m}^{-1} \quad (29)$$

Hvilket giver følgende teoretiske halveringstykkelser:

$$\text{Al}_{x_{\frac{1}{2}}} = \frac{\ln(2)}{20,061667\text{m}^{-1}} \approx 34,55\text{mm} \quad (30)$$

$$\text{Pb}_{x_{\frac{1}{2}}} = \frac{\ln(2)}{117,4725\text{m}^{-1}} \approx 5,905\text{mm} \quad (31)$$

Med den teoretiske og observerede værdi for absorptionskoefficienten, kan den pro-

⁴²Delacroix, Guerre, and Leblanc 2002, s. 114.

⁴³NIST 2010.

⁴⁴NIST 2004.

centvise afvigelse beregnes:

$$Al_{\text{afvigelse}} = \frac{15,12\text{m}^{-1} - 20,061667\text{m}^{-1}}{20,061667\text{m}^{-1}} \approx 24\% \quad (32)$$

$$Pb_{\text{afvigelse}} = \frac{86,73\text{m}^{-1} - 117,4725\text{m}^{-1}}{117,4725\text{m}^{-1}} \approx 26\% \quad (33)$$

Den observerede absorptionskoefficient for begge materialer afviger relativt meget fra den teoretiske, hvilket kan være grundet flere forskellige fejlkilder. Hvor det primært nok kan skyldes fejlmåling af pladernes tykkelser, da pladernes tykkelser varierede fra plade til plade, og var svære at måle eksakt. Selvom der er måleusikkerheder i forsøget, giver det stadig et meget godt indblik i forskellige materials beskyttelsesevner. Bly har en betydelig lavere halveringstid $x_{\frac{1}{2}}$, og en højere absorptionskoefficient μ , end aluminium, hvilket betyder at bly absorberer strålingen meget bedre end aluminium og er derfor oplagt at anvende fremfor aluminium til at beskytte mod stråling.

Forudsigelse af ustabile kerner henfald

Som tidligere forklaret, så vil barium-137 i exciteret tilstand henfalde til en lavere liggende energitilstand (5). Det er umuligt at vide hvornår en enkelt ustabil kerne henfalder⁴⁵, men ud fra henfaldsloven (8), ses det at sandsynligheden for at en enkelt kerne henfalder er konstant.

Hvis der måles flere gange på en radioaktiv kilde (som består af mange kerner N) over tidsintervaller som er meget kortere end halveringstiden, vil tællertallet T , variere fra måling til måling. Det viser sig at antallet af henfald pr. tidsinterval følger den diskrete poissonfordeling, da sandsynligheden for henfald er konstant, som har sandsynlighedsfunktionen:

$$P(n) = \frac{\lambda^n \cdot e^{-\lambda}}{n!} \quad (34)$$

Hvor λ er middelværdien og $n \in \mathbb{Z}$ er antallet af henfald i tidsintervallet. For en poissonfordeling gælder det at:

$$\sigma = \sqrt{\lambda} \quad (35)$$

⁴⁵Cappellaro 2023, kap 1.3 (s. 15).

Hvor σ er spredningen. Poissonfordeling er skæv, men for $\lambda \geq 30$, så er den næsten symmetrisk, og kan tilnærmes normalfordelingen:

$$P(x) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{-\frac{(x-\lambda)^2}{2\sigma^2}} \quad (36)$$

Hvor $x \in \mathbb{R}$ nu er kontinuer i stedet for diskrete heltal $n \in \mathbb{Z}$, som i poissonfordeling. Ved måling af tælleallet fra cæsium-137 hvert sekund i en periode på 3200s, fås en frekvensdiagram, som tydeligt forstiller en normalfordeling:

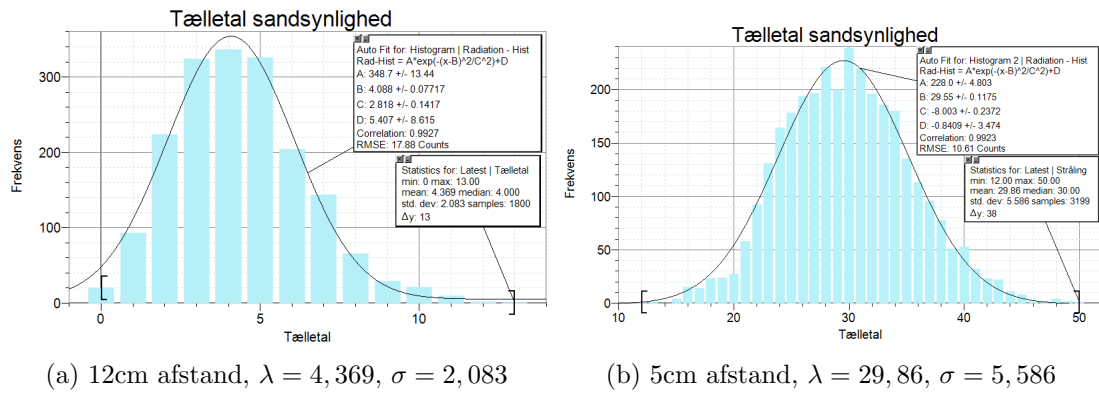


Figure 4: Frekvensdiagram ved forskellige afstande

Med Logger Pro⁴⁶, laves en normal kurvetilpasning, og det ses at afvigelsen mellem forholdet på middelværdien λ , og spredningen σ (35), er meget lille:

$$a_{\text{afvigelse}} = \frac{2,083 - \sqrt{4,369}}{\sqrt{4,369}} \approx 0,34\% \quad (37)$$

$$b_{\text{afvigelse}} = \frac{5,586 - \sqrt{29,86}}{\sqrt{29,86}} \approx 2,2\% \quad (38)$$

Hvor $a_{\text{afvigelse}}$ og $b_{\text{afvigelse}}$ henholdsvis referere til figur 4a og 4b. Selvom man ikke kan vide hvornår en enkelt kerne henfalder, kan det konstateres at man ved hjælp af statistik, kan forudsige hvornår det mest sandsynligt sker (Se evt. **Bilag 1 - Sandsynlighed for tællelet mellem 40 og 60**).

⁴⁶Vernier 2021.

Barium-137 halveringstid

$^{137}_{56}\text{Ba}^*$ har en teoretisk halveringstid på $T_{\frac{1}{2}} = 153,12\text{s}$ ⁴⁷. Eksperimentet vil forsøge at bekræfte halveringstiden. Der startes med at måle baggrundstrålingen i nogle minutter. Derefter tilføjer vi udtrækningsvæksen, saltsyre opblandet med NaCl, til en minigenerator som indeholder cæsium-137 kilden. Dette udtrækker barium-137 i isomér (exciteret) tilstand og det lander i en kapsel, med et GM-rør ved siden af. GM-røret måler dermed tællertallet i et interval på 10s, og følgende graf fås:

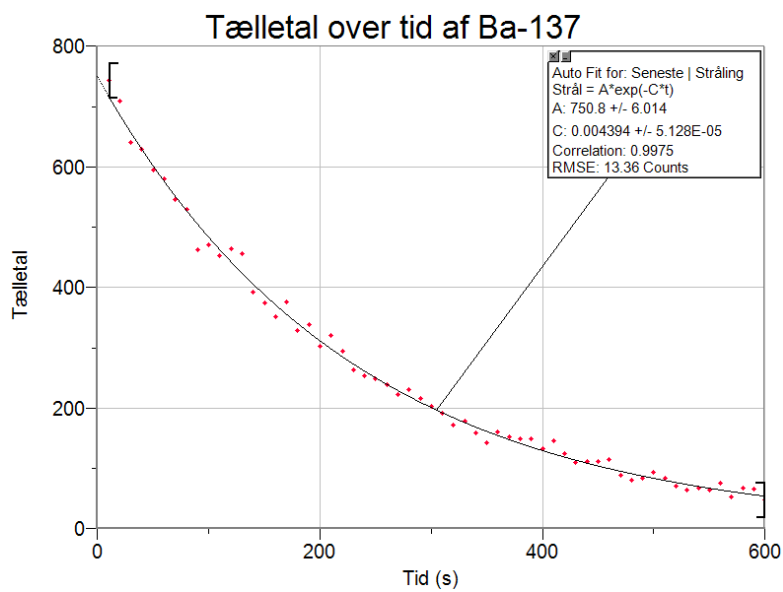


Figure 5: Aktiviteten af exciteret barium-137 falder over tid, $R^2 = 0,9976$

Det kan aflæses fra Logger Pros eksponentielle kurvetilpasning at:

$$T(t) = 750,8 \cdot e^{-0,004394t} \quad (39)$$

Ud fra udledningen af halveringstiden (9), kan det ses at den observerede halveringstid er:

$$T_{\frac{1}{2}} = \frac{\ln(2)}{k} = \frac{\ln(2)}{0,004394\text{s}^{-1}} \approx 157,7485618\text{s}$$

⁴⁷Audi et al. 2004, s. 76.

Hvilket giver en procentvis afvigelse på:

$$\text{Ba}_{\text{afvigelse}} = \frac{\frac{\ln(2)}{k} = \frac{\ln(2)}{0,004394s^{-1}} - 153,12s}{153,12s} \approx 3,02\%$$

Analyse af fejl korrigeringskoder

Repetition Koder

Den simpleste og mest naive fremgang til at lave en FEC, er bare at gentage sin besked flere gange. På den måde kan modtageren tage hvad der mest sandsynligt er rigtigt. ($n = 3, k = 1$) repetition koden er en lineær kode, som sender sin besked 3 gange. For denne kode er generator matricen, $G = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$. Antag man har koden $t = 010$, som skal enkodes til c . Der enkodes k bits ad gangen, med brug af generator matricen til at generere kodeordet (16):

$$\begin{aligned} t_1 \cdot G &= \begin{bmatrix} 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = 000 \\ t_2 \cdot G &= \begin{bmatrix} 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = 111 \\ t_3 \cdot G &= \begin{bmatrix} 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = 000 \\ c &= 000 \ 111 \ 000 \end{aligned}$$

Hvis vi nu sender c over en data kanal, hvor fejlen $e \in \mathbb{F}_2^n$ bliver tilført, hvilket medfører at modtageren modtager et kodeord $z = c + e \pmod{2}$. Hvordan retter modtageren fejlen, og får c tilbage?

Jf. minimum hamming distance formelen (50) (Se evt. **Bilag 2 - Uddledning af minimum hamming distance**):

$$d = \min_c wt(c) = wt(111) = 3$$

Hvilket betyder at den kan detektere $3 - 1 = 2$ fejl (13) og korrigere $\lfloor \frac{3-1}{2} \rfloor = 1$ fejl (14).

Vi antager derfor at der maksimalt forekommer 1 bit-flip: $wt(e) = 1$, ergo hamming vægten af fejlen tilføjet til c er maks 1. Hvis $e = 010$ og påvirker transmission af $c_2 = 111$,

så vil den modtagne besked blive:

$$\begin{aligned} z_2 &= c_2 \oplus e \\ &= 111 \oplus 010 \\ &= 101 \end{aligned}$$

Ergo den modtagne besked bliver $z = 000 \ 101 \ 000$. Modtageren har brug for H for at kunne rette fejlen, som fås ud fra forholdne mellem G og H (15) (20):

$$\begin{aligned} H &= [S^T | I] \\ &= \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \end{aligned}$$

Ved multiplikation af de modtagne kodeord fås:

$$\begin{aligned} Hc_1^T &= Hc_3^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ Hc_2^T &= \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{aligned}$$

Resultatet for c_2 er syndrom vektoren. Den bit hvor der er forekommet et bit-flip, svarer til nummeret af den kolonne som er lig med syndrom vektoren. I dette tilfælde, svarer syndrom vektoren til H s anden kolonne, hvilket betyder at det er anden bit, hvor der er forekommet et bit-flip. Modtageren kan nu rette fejlen ved at flippe bittet tilbage:

$$c_2 \oplus 010 = 101 \oplus 010 = 111$$

Modtageren har nu den korrigerede sekvens af kodeord, $000 \ 111 \ 000$. I reelle implementationer bliver der tit brugt en opslagstabel som kobler syndromer og fejl, hvilket gør man hurtigt kan rette fejlen⁴⁸ (Se evt. **Bilag 4 - Opslagstabel for fejl korrigerig**).

⁴⁸Beutelspacher and Rosenbaum 1998, s. 190.

Hamming Koder

En hamming kode er en (n, k) lineær kode, hvor minimum distancen mellem enhver to kodeord er $d = 3$ ⁴⁹. Dette betyder at den kan korrigere $\lfloor \frac{3-2}{2} \rfloor = 1$ fejl ud fra formel (14). Hamming (7,4) koden tilføjer 3 redundans bits $p_0 \ p_1 \ p_2$, til en 4 bit besked $x_3 \ x_2 \ x_1 \ x_0$. Den genererede kodeord er på formen $p_0 \ p_1 \ x_3 \ p_2 \ x_2 \ x_1 \ x_0$.

Redundans bitsne er defineret af følgende redundans ligninger^{50 51}:

$$p_0 = x_3 \oplus x_2 \oplus x_0 \quad (40)$$

$$p_1 = x_3 \oplus x_1 \oplus x_0 \quad (41)$$

$$p_2 = x_2 \oplus x_1 \oplus x_0 \quad (42)$$

Ved enkodning af f.eks. 1010 vil kodeordet blive 1011010 baseret på (40), (41) og (42). Generator matricen for denne kode, som enkoder en kode på 4 bits til et kodeord på 7 bits er defineret som:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (43)$$

Man kan se at den følger formaten som beskrevet af (15), ved at de første 4 kolonner er identitets matricen I . Ved enkodning af bitsne ud fra formel (16):

$$\begin{bmatrix} x_3 & x_2 & x_1 & x_0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x_3 & x_2 & x_1 & x_0 & p_0 & p_1 & p_2 \end{bmatrix} \pmod{2}$$

Altså identitets matricen I , af G kopierer x_3, x_2, x_1, x_0 over i resultatet og hver kolonne af S korresponderer henholdsvis til en af redundans ligningerne (40), (41) eller (42). Dette er fordi at alle elementerne i G er i det endelige legeme \mathbb{F}_2 , hvilket svarer til redundans ligningernes \oplus operator. Efter alle redundans bitnse er beregnet rykkes bitsne rundt så

⁴⁹Ziener and Tranter 2006, s. 644.

⁵⁰Eigenchris 2019, ep. 1.

⁵¹Pavert 2011, s. 19.

de kommer på formen $p_0p_1x_3p_2x_2x_1x_0$. Ligesom repetition koden har hamming koden en parity-check matrice H , som ud fra (20) er:

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \quad (44)$$

For dekodning af hamming koden gør man ligesom i repetition koden. Given hamming ($n = 7, k = 4$) koden $C \subset \mathbb{F}_2^n$, hvor afsenderen har transmitteret $c \in C$, som er blevet påvirket af fejlen $e \in F_2^n$ hvor $wt(e) = 1$ - der er 1 fejl. Vil den modtagne kodeord være $z = c + e \pmod{2}$. Det er oplagt at $z \notin C$, da $wt(e) \neq d$. For eksempel hvis $c = 1011010$ så rykkes der rundt på bitsne så den kommer på formen $x_3x_2x_1x_0p_0p_1p_2$, så $c = 1010101$, $e = 0100000$, så er $z = c + e \pmod{2} = 1010101 \oplus 0100000 = 1110101$. Derfor beregnes syndrom vektoren med formel (18):

$$Hz^T = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \pmod{2}$$

Syndrom vektoren svarer til H s anden kolonne, hvilket betyder at fejlen forekom i andet bit. Derfor kan fejlen korrigeres ved at flippe bittet tilbage: $z + e = c \pmod{2} = 1110101 \oplus 0100000 = 1010101$.

Reed-Solomon kode

Reed-Solomon (RS) koder bliver brugt en række steder i dagligdagen, som i QR koder, CD/DVD'er⁵² og harddiske⁵³. Den første praktiske brug af koden var i Voyager 2 rummissionen til Uranus og Neptun i 1977⁵⁴ ⁵⁵. Det endelige legeme for RS-koder er tit

⁵²Czynszak 2011, s. 6.

⁵³Moulton 2007.

⁵⁴Huffman and Pless 2003, s. 613.

⁵⁵McEliece and Swanson 1993, s. 6.

$\mathbb{F}_2^{2^8}$. Enkodning af en RS-kode består af at generere en irreduktibel generator polynomie (den kan ikke faktoreres) $g(x)$. Generator polynomiet er produktet af faktorene $(x - \alpha^n)$ hvor n er antallet af redundans bytes, hvilket giver:

$$\begin{aligned} g(x) &= \prod_{i=0}^{n-1} (x - \alpha^i) \\ &= (x - \alpha^0)(x - \alpha^1) \dots (x - \alpha^{n-1}) \end{aligned} \tag{45}$$

α er en primitiv element i det endelige element $\mathbb{F}_2^{2^8}$. Efter genereringen af $g(x)$, kan kodeordet $c(x)$ beregnes ved dividering mellem $t(x)$ og $g(x)$ over det endelige legeme $\mathbb{F}_2^{2^8}$ med formel (22). Dekodning af en RS-kode er drastisk sværere end enkodning. RS codec programmet der vil blive implementeret, vil understøtte erasure korrigering, og ikke en fuldstændig dekodnings algoritme, da det overgår rammerne for dette projekt. Erasure korrigering kan korrigere fejl, hvis man oplyser hvilke byte(s) der er fejl i. Den grundlæggende fremgangsmetode består af at:

- Beregn syndromet af den modtagne besked. hvis syndromet er 0 - ingen fejl, ellers er der en fejl.
- Beregn erasure locator polynomie fra syndromet med Berlekamp-Massey algoritmen.
- Beregn erasure evaluator polynomie fra syndromet og erasure locator polynomie.
- Beregn korruptions polynomiet, som indeholder fejlen, med Forney algoritmen og de 3 polynomier.
- Til sidst kan korruptions polynomiet trækkes fra (det samme som at plusse i $\mathbb{F}_2^{2^8}$) den modtagne kodeord, og den er nu korrigeret.

Se gerne **Bilag 7 - Detaljeret process af erasure korrigering**, for en gennemgang af hvad hvert skridt betyder og gør.

Design & Implementation

Der skal udvikles to codecs - et program som kan encode og decode data - for en hamming- og en RS kode. Hvert codec skal kunne:

- Tage imod bruger input.
- Enkode dataet til et kodeord med fejl korrigeringskoden.
- Dekode kodeordet tilbage til dataet.
- Kunne dekode et kodeord hvor 1 bit-flip er forekommet (For RS-koden er positionen oplyst).

Da fejl korrigeringskoder arbejder på rå data, er det oplagt at anvende et lavt niveau sprog, for at forbedre effektiviteten. I lavt niveau sprog, som C/C++, er det påkrævet at man selv holder styr hukommelse, hvilket betyder at det er meget tydeligt at se hvad der forgår uden alt for mange ting som sker bag scenen. Codecs har som regel en simpel facade med en `encode()` og `decode()` funktion. Dette bygger på facade design mynsteret, hvilket giver en interface som er nemmere at bruge⁵⁶. Selve facaden er komponeret af underlægge sub-rutiner. F.eks. så består `encode()` funktionen af underopgaver som den skal løse for at få resultatet. Baseret på at fejl korrigeringsalgoritmer arbejder meget på at transformere data og lave matematiske udregner, vil der bruges en *procedural* programmerings paradigme. Procedural programmering består af at dekomponere et stort problem (som f.eks `encode()` el. `decode()`), ned til mindre problemer i en funktionel dekomponeringsproces, indtil at problemet kan løses⁵⁷. Det er derfor et oplagt paradigme at anvende, fremfor f.eks. Objekt Orienteret Programmering (OOP), som i nogle tilfælde kan introducere unødvendig kompleksitet⁵⁸, og være et performance problem⁵⁹. Der vil altså blive brugt C++ med et procedural paradigme.

Hamming codec

Der udvikles en ($n = 7, k = 4$) Hamming codec. Det betyder at inputtet består af k bits og det enkodet kodeord består af n bits. Den består af en enkoder, som tager en bytesekvens og deler hver byte op i 2, da $1\text{ byte} = 8\text{ bits} \implies 2 \cdot 4\text{ bits} = 1\text{ byte}$. Da $2n > 1\text{ byte}$, bliver resultat bytesekvensen 2 gange større end inputtet. Den sidste bit af kodeordet vil gå til spilde og ikke blive brugt. Det betyder at hvis `encode()`, får en 4byte input ind, så spytter den en 8byte kodeord ud og vice-versa med `decode()`.

⁵⁶Gamma et al. 2009, s. 185.

⁵⁷Stevenson 2013, s. 1.

⁵⁸MacHaffie 1993, s. 9.

⁵⁹Wingqvist, Wickström, and Memeti 2022, kap. II.B.

Enkodning

En af underopgaverne består som sagt af at tage en 4bit besked ind, og encode den til et 8 bit kodeord, hvor den sidste bit ikke bliver brugt. Dette giver en funktions prototype på: `uint8_t encode(uint8_t data)`. `uint8_t` er en primitiv datatype som består af 1byte og er derfor perfekt. I `data` variabelen er det kun de første 4bits som bliver brugt. Standarden for enkodning af 4bits $x_0x_1x_2x_3$ er at indsætte redundans bitsne ind i mellem på formen:⁶⁰.

$$p_0p_1x_3p_2x_2x_1x_0 \quad (46)$$

```

1 uint8_t encode_hamming7_4chunk(uint8_t code) {
2     // Go from 0 0 0 0 x3 x2 x1 x0 -> 0 0 0 x3 x2 x1 x0 0
3     code <<= 1;
4
5     // Swap x3 and p2: 0 0 0 x3 x2 x1 x0 -> 0 0 x3 0 x2 x1 x0 0
6     uint8_t tmp = ((code >> 5) & 1) ^ ((code >> 4) & 1);
7     tmp = (tmp << 4) | (tmp << 5);
8     code ^= tmp;
9
10    uint8_t x3 = code & (1 << 5);
11    uint8_t x2 = code & (1 << 3);
12    uint8_t x1 = code & (1 << 2);
13    uint8_t x0 = code & (1 << 1);
14
15    uint8_t p0 = (x3 << 2) ^ (x2 << 4) ^ (x0 << 6);
16    uint8_t p1 = (x3 << 1) ^ (x1 << 4) ^ (x0 << 5);
17    uint8_t p2 = (x2 << 1) ^ (x1 << 2) ^ (x0 << 3);
18
19    return code | p0 | p1 | p2;
20 }
```

Listing 1: Enkodning af en chunk af 4bit

Funktionen starter med at lave en bitwise skift til venstre på koden, hvilket får `code` på formen $000x_3x_2x_1x_0$ (l. 3). Herefter ombyttes x_3 og p_2 hvilket gør så koden kommer på standard form (46) (ll. 6-8). Ombytningen virker ved at xor (^symbolet) p_2 og x_3 i det første bit af byen (Least significant bit) (l. 6). Derefter rykkes resultatet af xoren

⁶⁰Jeffrey 2005, s. 1.

tilbage til bit positionerne (l. 7), og koden bliver xored med bit positionerne (l. 8). Nu er koden på standard form, og redundans bitsne skal beregnes. Først isoleres bitsne for henholdsvis x_3, x_2, x_1 og x_0 , ved en bitwise AND (& symbolet) mellem koden og en bit mask for bit positionen af enhver x i koden (ll. 10-13). Nu har vi alle værdier forberedt til at kunne anvende hamming (7,4) kodens redundans ligninger og beregne hver redundans bit (se (40), (41) og (42)). Hvert redundans bit bliver beregnet ved at rykke hver data bit x_i til bit positionen af hvor redundans bitten skal ende op i kodeordet. Efter at enhver data bit x_i er på samme bit position, beregnes xoren mellem dem, som i redundans ligninger (ll. 15-17). Til sidst, når alle redundans bit er beregnet og befinder sig på de rigtige bit positioner bliver bitwise OR (| symbolet), brugt til at kombinere hvert enkelt redundans bit og data bit sammen i 1 byte. Chunken er nu enkodet. Denne funktion kan enkode 4 bits, hvilket betyder at vi skal bruge endnu en funktion som kan splitte en byte sekvens op, og enkode dem i chunks:

```
1 int encode_hamming7_4(uint8_t *buffer, uint8_t *codeword, unsigned int
   size) {
2     for (int i = 0; i < size; i++) {
3         uint8_t code = buffer[i];
4
5         // Encode lower part
6         codeword[i * 2] = encode_hamming7_4chunk(code);
7
8         // Encode upper part
9         codeword[i * 2 + 1] = encode_hamming7_4chunk(code >> 4);
10    }
11    return 0;
12 }
```

Listing 2: Enkodning af en bytesekvens

Dette er facade funktionen. Den tager imod en pointer til en **buffer**, som indeholder en bytesekvens, og en pointer til en buffer som er allokeret til at rumme det enkodede kodeord **codeword**. Det gælder her at **codeword** bufferen, skal være dobbelt så stor som input kode bufferen **buffer**. Til sidst, tager den imod størrelsen på **buffer** så den ved hvor mange bytes, den skal enkode fra input bufferen. Den starter med en forlykke over **size** - antallet af bytes der skal enkodes (l. 2). Nu bruges sub-rutinen som vi lavede før **encode_hamming7_4chunk**, til at enkode hver byte ad gangen. Da **encode_hamming7_4chunk** enkoder 4bits, bruger vi den til at enkode de første 4bits af

code (l. 6), og derefter de sidste 4bits, ved at skifte hele koden 4 bit positioner til højre (l. 9). De enkodede bytes skrives til `codeword` bufferen.

Dekodning

Dekodning er processen af at tage kodeordet, korrigere eventuelle fejl, og returnere den oprindelige besked. Da den enkodede repræsentation fylder dobbelt så meget, bliver vi nødt til at dekode en chunk på 1 byte to gange for hver information byte. Det er derfor oplagt at lave en sub-rutine til at dekode en chunk:

```
1 uint8_t decode_hamming7_4chunk(uint8_t codeword) {
2     // Check for valid codewords
3     if (auto it = codewordToDecoded.find(codeword);
4         it != codewordToDecoded.end()) {
5         return it->second;
6     } else {
7         // Codeword is not valid - try and fix it
8         uint8_t p0 = (codeword >> 7) & 1;
9         uint8_t p1 = (codeword >> 6) & 1;
10        uint8_t x3 = (codeword >> 5) & 1;
11        uint8_t p2 = (codeword >> 4) & 1;
12        uint8_t x2 = (codeword >> 3) & 1;
13        uint8_t x1 = (codeword >> 2) & 1;
14        uint8_t x0 = (codeword >> 1) & 1;
15
16        // calculate the syndrome bits
17        uint8_t p0_prime = x3 ^ x2 ^ x0;
18        uint8_t p1_prime = x3 ^ x1 ^ x0;
19        uint8_t p2_prime = x2 ^ x1 ^ x0;
20
21        // error in x3
22        if (p0 != p0_prime && p1 != p1_prime && p2 == p2_prime)
23            codeword ^= 1 << 5;
24        // error in x2
25        else if (p0 != p0_prime && p1 == p1_prime && p2 != p2_prime)
26            codeword ^= 1 << 3;
27        // error in x1
28        else if (p0 == p0_prime && p1 != p1_prime && p2 != p2_prime)
29            codeword ^= 1 << 2;
30        // error in x0
31        else if (p0 != p0_prime && p1 != p1_prime && p2 != p2_prime)
```

```

32         codeword ^= 1 << 1;
33         uint8_t decoded = 0;
34         // write data bits to output positions 0 0 0 0 x3 x2 x1 x0
35         decoded |= ((codeword >> 5) & 1) << 3; // x3
36         decoded |= ((codeword >> 3) & 1) << 2; // x2
37         decoded |= ((codeword >> 2) & 1) << 1; // x1
38         decoded |= ((codeword >> 1) & 1);      // x0
39         return decoded;
40     }
41 }

```

Listing 3: Dekodning af en chunk på 1 byte

Da der kun er $2^n \implies 2^4 = 16$ forskellige kodeord, er det nemmest (og hurtigst), at have en opslagstabel som korresponderer kodeord til den dekode data:

```

1 std::unordered_map<uint8_t, uint8_t> codewordToDecoded = {
2     {0b00000000, 0}, {0b11010010, 1}, {0b01010100, 2},
3     {0b10000110, 3}, {0b10011000, 4}, {0b01001010, 5},
4     {0b11001100, 6}, {0b00011110, 7}, {0b11100000, 8},
5     {0b00110010, 9}, {0b10110100, 10}, {0b01100110, 11},
6     {0b01111000, 12}, {0b10101010, 13}, {0b00101100, 14},
7     {0b11111110, 15},
8 };

```

Listing 4: Pre-komputeret opslagstabel for kodeord til kode

Dekode funktionen kan så tjekke efter kodeordet i opslagstabellen i en gennemsnitlige kompleksitet af $O(1)$ ⁶¹, og direkte returnere den dekode data (ll. 3-5). Hvis kodeordet derimod ikke er i opslagstabellen, ved vi at der er sket en fejl. Vi kan dermed starte med at isolere alle bits for sig, og gemme dem i hver sin variabel, så det er lettere at arbejde med (ll. 8-14). Dette bliver gjort ved at rykke hvert bit ned på den første bit position (least significant byte), og derefter bitwise AND med en bitmask på 1, hvilket sætter alle andre bit positioner til 0⁶². Derefter beregnes redundans bitsne for kodeordet baseret på dataet med redundans ligningerne (40), (41) og (42) (ll. 17-19). Hvis de beregnede redundans bits ikke passer med redundans bitsne i det enkode kodeord, må der være sket en fejl i en af de information bits som var med til at beregne redundans bitten. F.eks. hvis redundans bit p_0 og p_1 ikke passer, men p_2 passer, må der have været sket en

⁶¹cppreference.com 2021.⁶²Anderson 2005.

fejl i x_3 , da x_3 er den eneste som ikke er med i redundans ligning for p_2 (42), men er med i p_0 og p_1 (l. 22). Fejlen korrigeres så ved at flippe bittene på x_3 bit positionen tilbage (l. 23). Det samme sker henholdsvis for x_2, x_1 og x_0 . Til sidst bliver hver information bit rykket hen på plads, så de fylder de 4 første bits af byten, og den dekode data bliver returneret.

```
1 void decode_hamming7_4(uint8_t *codewordBuf, uint8_t *outBuffer,
2                         unsigned int size) {
3     for (int i = 0; i < size; i++) {
4         uint8_t lower = decode_hamming7_4chunk(codewordBuf[i]);
5         uint8_t upper = decode_hamming7_4chunk(codewordBuf[++i]) << 4;
6         outBuffer[i] = lower | upper;
7     }
8 }
```

Listing 5: Dekodning facade funktion

Denne funktion er facaden til dekodning. Den tager en pointer til en bytesekvens af kodeord `codewordBuf`, samt dens længde `size`, og en pointer til en buffer, hvor den dekode data skal skrives til. Den dekode henholdsvis den første del af dataet (l. 4), og derefter de øvre 4 bits (l. 5). Til sidst kombineres de dekode oppe og nedre dele og skrives til output bufferen `outBuffer` (l. 6). Se **Bilag 10 - Test af hamming koden**, for test af programmet.

Reed-Solomon codec

Det er ude for opgavens rammer, at detaljeret forklare koden bag implementationen (der ville nemt blive brugt over 10 sider). I stedet forklares et overfladisk flowdiagram, over encode og decode facade funktionerne, som forklarer algoritmen på et højt niveau.

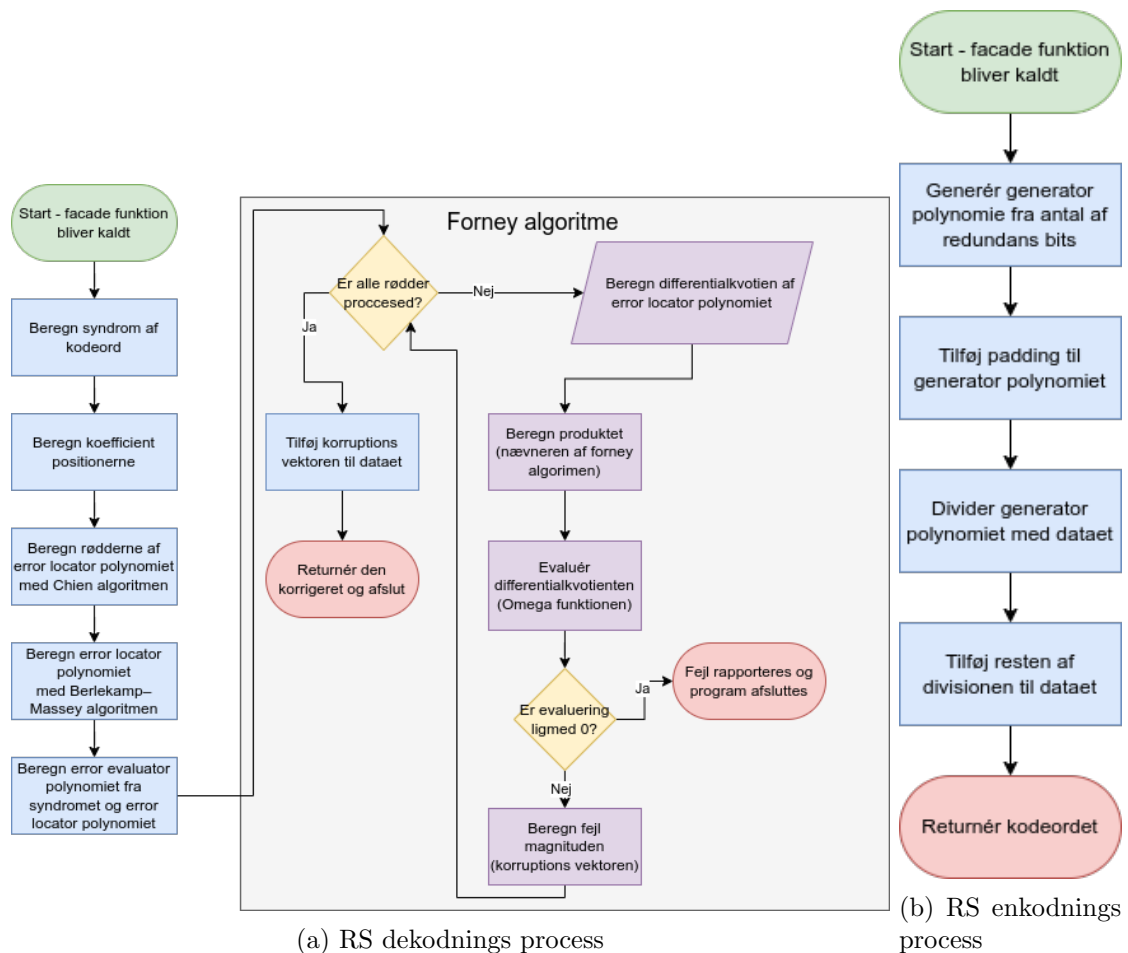


Figure 6: Flowdiagram over RS dekodning og enkodning

Dekodningen består altså af at beregne og transformere en masse data, evaluere polynomier og beregne differentialkvotienter af polynomier. Slutresultatet er korruptions polynomiet som trækkes fra kodeordet, hvilket giver det korrigerede kodeord. Mange af disse skridt er de samme som beskrevet i **Bilag 7 - Detaljeret process af erasure korrigering**. For en god forståelse anbefales det at læse kapitel 7.4, 7.5 og 7.6⁶³, og Wikiversity's "Reed-Solomon codes for coders"⁶⁴. Se **Bilag 9 - Kildekode af RS codec**, for kildekoden.

⁶³Blahut 2003, s. 193-210.

⁶⁴Reed-Solomon codes for coders 2023.

Diskussion

Det er blevet set at problemet med bit-fejl, kan løses på mange forskellige måder, både med brug af software/hardware i form af fejl korrigeringskoder, og fysisk beskyttelse mod stråling. Analysen af materials halveringstykkelser har vist at tungere materialer som bly beskytter drastisk bedre i forhold til lettere materialer som aluminium. Dette betyder dog ikke at bly er det rigtige materiale at bruge til alle formål, da det også har nogle ulemper. Blys densitet er hvad der gør beskyttelsesevnen god, men det betyder også at ens computer systemer og elektroniske kredsløb bliver tungere, som kan være upraktisk i situationer hvor massen er en kritisk faktor, som f.eks. i raketter. I situationer hvor de elektroniske kredsløb er stationære giver det god mening at anvende. Begge materialer undersøgt har altså hver deres fordele og ulemper, og det bedste materiale afhænger meget af situationen.

Fejl korrigeringskoder har vist sig at være til stede i rigtig mange ting. De er tit anvendt i situationer hvor der ikke er et fysisk lag af beskyttelse, som i trådløs kommunikation. Der er rigtig mange metoder, afhængig om ens formål er at detektere fejl eller også at kunne korrigere fejl. Hamming koden og RS koden, løser problemet på forskellige måder. Hamming koden er klart simplere at implementere, men også svagere idet at den maksimalt kan korrigere 1 fejl. Hamming koden er derfor oplagt i situationer hvor at regnekraften er ekstremt begrænset. RS koden har vist sig at være drastisk mere kraftfuld idet at den kan korrigere op til mange fejl, men det introducerer også endel kompleksitet, specielt i dekodningsprocessen. I visse embedded tilfælde bliver disse FEC algoritmer tit implementeret i hardware⁶⁵. Da enkodningsprocessen for RS-koden er relativ simpel og kan gøres med accelereret hardware specifikt designet til det, var RS-koden en pragmatisk og effektiv kode at anvende i Voyager 2 rummissionen.

Det er altså ikke så simpelt som bare at inkapsulere sit elektronik i et tykt bly skjold, og bruge den kraftigste fejl korrigeringsalgoritme. Der er mange andre faktorer, som har indflydelse på henholdsvis hvilket materiale der er egnet, samt hvilken fejl korrigeringsalgoritme der er egnet til den specifikke situation.

⁶⁵Sebastian and Bonna 2012.

Konklusion

Der er blevet vist at bit-flip er en kritisk fejl for digitale computer systemer, som Richard Hamming beskriver i sit papir omkring fejl korrigeringskoder⁶⁶. Der blev fundet ud af at atomkerner kan blive radioaktive, og udsende stråling ved henfald, som statistisk kan forudsiges. Forskellige materials beskyttelsesevne er blevet undersøgt i form af deres halveringstid, og det viser sig at tungere materialer som bly beskytter bedre end lettere materialer som aluminium. Der er stadig en række praktiske forhold som gør at bly ikke altid er det mest egnede materiale. Det er blevet vist at fejl detektionskoder kan bruges til at detektere fejl og bede om gen-transmission i feedback kanaler. Derudover er der blevet undersøgt mere sofistikeret FEC algoritmer, som Hamming koder og RS-koder. Det viser sig at RS-koder, er mere kraftfulde end hamming koder, og bliver brugt rigtigt mange steder i dag, og var den primære fejl korrigeringskode brugt i NASAs Voyager 2 rummission. Der er blevet designet og implementeret to FEC codecs, som tydeligt demonstrerer hvordan man kan anvende fejl korrigeringskoder i software til at bevare integriteten af data, selvom det bliver korrumperet under transmission.

Der er rigtig mange forskellige fremgangsmetoder til at beskytte integriteten af data på. Den egnede løsning til data korruption varierer meget i forhold til hvilken et miljø dataet befinder sig i, samtidig med ens formål. Er der f.eks. kort distance mellem afsender og modtager? så er en simpel fejl detektionskode sammen med en feedback kanal typisk nok. Er der derimod langt distance mellem afsender og modtager, som set i Voyager 2 missionen? så er en fejl korrigeringskode oplagt at anvende.

⁶⁶Hamming 1950, s. 147.

Kildeliste

- Hamming, Richard Wesley (1950). *Error Detecting and Error Correcting Codes*. URL: <https://zoo.cs.yale.edu/classes/cs323/doc/Hamming.pdf>. Besøgt: 11 dec 2023.
- Glover, Neal and Trent Dudley (1991). *Fundamentals of Error-Correcting Codes*. URL: http://www.bitsavers.org/components/cirrusLogic/Practical_Error-Correction_Design_For_Engineers_2ed_1991.pdf. Besøgt: 11 dec 2023.
- MacHaffie, Scott Andrew (1993). *Difficulties Experienced Procedural Programmers Encounter When Transferring to an Object-oriented Programming Paradigm*. URL: https://pdxscholar.library.pdx.edu/cgi/viewcontent.cgi?article=5692&context=open_access_etds. Besøgt: 16 dec 2023.
- McEliece, Robert J. and Laif Swanson (1993). *Reed-Solomon Codes and the Exploration of the Solar System*. URL: <https://dataverse.jpl.nasa.gov/dataset.xhtml?persistentId=hdl:2014/34531>. Besøgt: 15 dec 2023.
- Beutelspacher, Albrecht and Ute Rosenbaum (1998). *Projective Geometry: From Foundations to Applications*. URL: <https://www.maths.ed.ac.uk/~v1ranick/papers/beutel.pdf>. Besøgt: 10 dec 2023.
- Delacroix, Guerre, and Leblanc (2002). *RADIONUCLIDE AND RADIATION PROTECTION DATA HANDBOOK 2002*. URL: <https://documents.manchester.ac.uk/display.aspx?DocID=26899>. Besøgt: 13 dec 2023.
- Wang, Charles, Dean Sklar, and Diana Johnson (2002). *Forward Error-Correction Coding*. URL: <https://web.archive.org/web/20120314085127/http://www.aero.org/publications/crosslink/winter2002/04.html>. Besøgt: 11 dec 2023.
- Blahut, Richard E. (2003). *Algebraic Codes for Data Transmission*. Cambridge University Press. ISBN: 9780521553742. URL: https://portal.sibadi.org/pluginfile.php/126471/mod_folder/content/0/Blahut%20Algebraic%20codes%20for%20data%20transmission.pdf?forcedownload=1. Besøgt: 18 dec 2023.
- Huffman, W. Cary and Vera Pless (2003). *Fundamentals of Error-Correcting Codes*. URL: <https://doc.lagout.org/Others/Information%20Theory/Coding%20Theory/Fundamentals%20of%20Error-Correcting%20Codes%20-%20W.%20Cary%20Huffman.pdf>. Besøgt: 10 dec 2023.
- Mackay, David J. C. (2003). *Information Theory, Inference and Learning Algorithms*.

- Marks, Roger B. (2003). *IEEE Standard 802.16 for Global Broadband Wireless Access*. URL: https://www.ieee802.org/16/docs/03/C80216-03_14.pdf. Besøgt: 09 dec 2023.
- Audi, G. et al. (2004). *The NUBASE evaluation of nuclear and decay properties*. URL: <https://hal.in2p3.fr/in2p3-00020241/document>. Besøgt: 18 dec 2023.
- NIST (2004). *X-Ray Mass Attenuation Coefficients*. URL: <https://physics.nist.gov/PhysRefData/XrayMassCoef/tab1.html>. Besøgt: 14 dec 2023.
- Anderson, Sean Eron (2005). *Bit Twiddling Hacks*. URL: <http://graphics.stanford.edu/~seander/bithacks.html>. Besøgt: 18 dec 2023.
- Danziger, P. (2005). *Linear Codes*. URL: <https://math.ryerson.ca/~danziger/professor/MTH108/Handouts/codes.pdf>. Besøgt: 09 dec 2023.
- Jeffrey (2005). *The Hamming [7,4,3] Code*. URL: <http://homepages.math.uic.edu/~leon/mcs425-s08/handouts/Hamming.pdf>. Besøgt: 17 dec 2023.
- Ziemer, Rodger and William H Tranter (2006). *Principles of communications: system modulation and noise*. John Wiley & Sons. Besøgt: 09 dec 2023.
- Feynman, Richard (2007). *The Feynman Lectures on Physics, Volume III*. URL: https://www.feynmanlectures.caltech.edu/III_toc.html. Besøgt: 12 dec 2023.
- Moulton, Scott A. (2007). *My Hard Drive Died!* URL: https://web.archive.org/web/20080202143103/http://www.myharddrivedied.com/presentations_whitepaper.html. Besøgt: 19 dec 2023.
- Gamma, Erich et al. (2009). *Design Patterns Elements of Reusable Object-Oriented Software*. ISBN: 0-201-63361-2. URL: <http://www.javier8a.com/itc/bd1/articulo.pdf>. Besøgt: 16 dec 2023.
- Mitchell, Gregory (2009). *Investigation of Hamming, Reed-Solomon, and Turbo Forward Error Correcting Codes*. URL: <https://apps.dtic.mil/sti/tr/pdf/ADA505116.pdf>. Besøgt: 11 dec 2023.
- Ryan, William E. and Shu Lin (2009). *Channel Codes Classic and Modern*. URL: https://d1.amobbs.com/bbs_upload782111/files_35/ourdev_604508GHLFR2.pdf. Besøgt: 10 dec 2023.
- Harrell, John (2010). *The Importance of ECC Memory in Your Substation Computer*. URL: https://lightriver.com/wp-content/uploads/2018/04/0004_importanceeccmemory_jh_20100716.pdf. Besøgt: 18 dec 2023.

- NIST (2010). *XCOM*. URL: <https://www.physics.nist.gov/PhysRefData/Xcom/html/xcom1.html>. Besøgt: 14 dec 2023.
- Yu, Fa-Xin et al. (2010). *Overview of Radiation Hardening Techniques for IC Design*. URL: <https://scialert.net/fulltext/fulltextpdf.php?pdf=ansinet/itj/2010/1068-1080.pdf>. Besøgt: 18 dec 2023.
- Czynszak, Szymon (2011). *Decoding algorithms of Reed-Solomon code*. URL: <https://www.diva-portal.org/smash/get/diva2:833161/FULLTEXT01.pdf>. Besøgt: 19 dec 2023.
- Pavert, León van de (2011). *REED-SOLOMON ENCODING AND DECODING*. URL: <https://www.theseus.fi/bitstream/handle/10024/32913/Reed-Solomon%20Encoding%20and%20Decoding.pdf>. Besøgt: 15 dec 2023.
- Sebastian, Aby and Kareem Bonna (2012). *Reed-Solomon Encoder and Decoder*. URL: <https://content.sakai.rutgers.edu/access/content/user/ak892/Reed-SolomonProjectReport.pdf>. Besøgt: 18 dec 2023.
- Stevenson, Joseph (2013). “Difference Between Object-oriented Programming and Procedural Programming Languages”. In: URL: <https://neonbrand.com/websites/development/procedural-programming-vs-object-oriented-programming-a-review/>. Besøgt: 16 dec 2023.
- Goemans, Michel (2015). *Linear Error-Correcting Codes*. URL: <https://math.mit.edu/~goemans/18310S15/Hamming-code-notes.pdf>. Besøgt: 10 dec 2023.
- Øhlenschläger, Erik (2018). *Grundlæggende fysik B*. ISBN: 9788702290233. URL: <https://grundlaeggendefysikb.systime.dk>. Besøgt: 12 dec 2023.
- Eigenchris (2019). *Error Correcting Codes (ECCs)*. URL: https://www.youtube.com/playlist?list=PLJHszsWbB6hqk0yFCQ0AlQtFzC1G9sf2_. Besøgt: 15 dec 2023.
- Michelsen, Kasper Grosman and Danni Pedersen (2019). *En verden af fysik B*. ISBN: 9788702290226. URL: <https://enverdenaffysikb.systime.dk>. Besøgt: 12 dec 2023.
- Poivey, Christian (2019). *RADIATION EFFECTS IN SPACE ELECTRONICS*. URL: <https://project-cms-rpc-endcap.web.cern.ch/rpc/ChambersandIntegration/Integration/RadiationExposure/TOTAL%20IONIZING%20AND%20NON-IONIZING%20DOSE%20RADIATION%20HARDNESS%20ASSURANCE.pdf>. Besøgt: 18 dec 2023.
- Overill, Richard E (2020). *Cosmic Rays: a Neglected Potential Threat to Evidential Integrity in Digital Forensic Investigations?* URL: <https://web.archive.org/web/>

- 20210901080013id_/https://kclpure.kcl.ac.uk/portal/files/138758922/WSDF_2020_invited_final.pdf. Besøgt: 12 dec 2023.
- Vepsäläinen, Antti P. et al. (Aug. 2020). “Impact of ionizing radiation on superconducting qubit coherence”. In: *Nature* 584.7822, pp. 551–556. ISSN: 1476-4687. DOI: 10.1038/s41586-020-2619-8. URL: <http://dx.doi.org/10.1038/s41586-020-2619-8>.
- cppreference.com (2021). URL: https://en.cppreference.com/w/cpp/container/unordered_map/find. Besøgt: 18 dec 2023.
- Holc, Per, Birgitte Merci Lund, and Jens Kraaer (2021). *Orbit A htx*. ISBN: 9788761696922. URL: <https://orbithtxa.systime.dk>. Besøgt: 12 dec 2023.
- Sena, Francisco Miguel Enxerto (2021). *Error-correcting linear codes*. URL: <https://www.math.tecnico.ulisboa.pt/~jnatar/MAGEF-20/Presentations/Error-correcting%20linear%20codes.pdf>. Besøgt: 10 dec 2023.
- Vernier (2021). *Logger Pro™ 3*. URL: <https://www.vernier.com/product/logger-pro-3/>.
- Wingqvist, David, Filip Wickström, and Suejb Memeti (2022). *Evaluating the performance of object-oriented and data-oriented design with multi-threading in game development*. URL: <https://www.diva-portal.org/smash/get/diva2:1732634/FULLTEXT01.pdf>. Besøgt: 16 dec 2023.
- Cappellaro, Paola (2023). *Introduction to Applied Nuclear Physics (Cappellaro)*. MIT OpenCourseWare. URL: [https://phys.libretexts.org/Bookshelves/Nuclear_and_Particle_Physics/Introduction_to_Applied_Nuclear_Physics_\(Cappellaro\)](https://phys.libretexts.org/Bookshelves/Nuclear_and_Particle_Physics/Introduction_to_Applied_Nuclear_Physics_(Cappellaro)). Besøgt: 14 dec 2023.
- Cook, Greg (2023). *Catalogue of parametrised CRC algorithms*. URL: <https://reveng.sourceforge.io/crc-catalogue/all.htm>. Besøgt: 09 dec 2023.
- Jensen, Michael, Klaus Marthinus, and Bernt Hansen (2023). *MAT A htx*. ISBN: 9788761693617. URL: <https://mathtxa.systime.dk>. Besøgt: 12 dec 2023.
- Qalculate (2023). *Qalculate*. URL: <http://qalculate.github.io/index.html>.
- Reed–Solomon codes for coders (2023). URL: https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders. Besøgt: 19 dec 2023.

Bilag

Bilag 1 - Sandsynlighed for tælleletal mellem 40 og 60

Lad os betragte at gamma-kilden står 5cm væk fra GM-røret, så modellen fra figur 4b bruges: $\lambda = 29,86$ og $\sigma = 5,586$. Dermed kan følgende normaliserede normal sandsynlighedsfunktionen konstrueres fra formel (36):

$$P(x) = \frac{1}{5,586\sqrt{2\pi}} \cdot e^{-\frac{(x-29,55)^2}{2 \cdot 5,586^2}} \quad (47)$$

For normaliserede gaussian (normal) sandsynlighedsfunktioner gælder

$$\int_{-\infty}^{\infty} P(x)dx = 1 \quad (48)$$

Og

$$P(a \leq x \leq b) = \int_a^b P(x)dx \quad (49)$$

Dermed, ved brug af CAS-værktøj (Qalculate)⁶⁷ løses den bestemte integrale:

$$\begin{aligned} P(40 \leq x \leq 60) &= \int_{40}^{60} P(x)dx \\ &\approx 3,47\% \end{aligned}$$

Bilag 2 - Udledning af minimum hamming distance

Given en lineær kode $C \subset F_2^n$, ved vi at en enhver lineær kombination af et kodeord er et andet defineret kodeord. Given to kodeord $c_1 \in C$ og $c_2 \in C$, så vil hamming distancen mellem dem være $d(c_1, c_2)$. Derfra ved vi, at ved at trække c_1 fra begge kodeord, er de stadig tilladte kodeord:

$$\begin{aligned} d(c_1 - c_1, c_2 - c_1) \\ d(0, c_2 - c_1) \end{aligned}$$

⁶⁷Qalculate 2023.

Jf. Lineær kombination reglen, er minimum distancen den laveste distance mellem 0 og alle tilladte kodeord:

$$d = \min_c d(0, c)$$

Da $d(0, c) = wt(c)$, må den minimum distance være den mindste hamming vægt for alle kodeord c i C :

$$d = \min_c wt(c), \forall c \in C, c \neq 0 \quad (50)$$

Hvor $wt(c)$ er hamming vægten af kodeordet c .

Bilag 3 - Mængde af fejl (7,4) Hamming kode kan detektere og korrigere

Jf. formlen for den minimum hamming distance (50), kan (7, 4) Hamming kodens distance beregnes til at være $d = 3$. Ud fra formlen for mængden af fejl der kan detekteres (13) og formlen for mængden af fejl der kan korrigeres (14), kan der udregnes følgende:

$$\begin{aligned} e_d &= d - 1 \\ &= 3 - 1 \\ &= 2 \end{aligned}$$

$$\begin{aligned} e_k &= \left\lfloor \frac{d-1}{2} \right\rfloor \\ &= \left\lfloor \frac{3-1}{2} \right\rfloor \\ &= 1 \end{aligned}$$

Ergo (7, 4) Hamming koden kan maksimalt detektere 2 fejl og korrigere 1 fejl.

Bilag 4 - Opslagstabel for fejl korrigerig

coset leader	syndrome
0000000	000
0000001	111
0000010	011
0000100	101
0001000	110
0010000	001
0100000	010
1000000	100

Figure 7: Opslagstabel som kobler syndrom til korrigerig

Antag at modtageren modtager kodeordet $c = 0010001$ og syndromet beregnes ud fra dens parity-check matrice H , til at være $s = 110$. Ved at kigge i opslagstabellen, ses det at syndromes korrespondere til $e = 0001000$. Ved summen af c og e , bliver fejlen korrigeret:

$$c \oplus e = 0010001 \oplus 0001000 = 0011001$$

Fejlen er nu korrigeret. e specificerer altså hvilke bits i c som skal flippes.

Bilag 5 - Energieneavu af Cæsium henfald

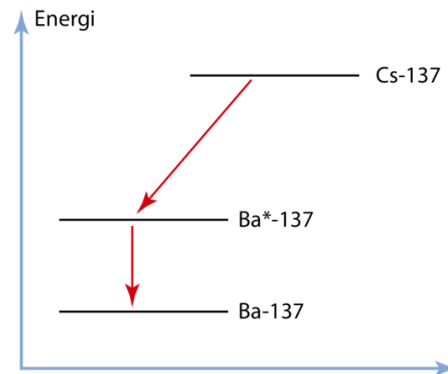


Figure 8: Energieneavu over Cæsium-137, Exciteret Barium-137 og Barium-137⁶⁸

Bilag 6 - Generering af alle kodeord ud fra generator polynomie

Givet en lineær cyklisk kode $C \subset \mathbb{Z}_2[x]/(x^6 - 1)$, og generator kodeordet $g = 100100$, så vil enhver lineær kombination og cyklisk skift være et tilladt kodeord. Generator kodeordet på polynomie form bliver til generator polynomiet:

$$g(x) = 1 + x^3$$

Dette giver følgende tabel, hvor produktet mellem x og $g(x)$ repræsenterer et cyklisk skift og addition er pga. koden er lineær.

Kodeord	polynomie	fra $g(x)$
000000	0	$0g(x)$
100100	$1 + x^3$	$g(x)$
010010	$x + x^4$	$xg(x)$
001001	$x^2 + x^5$	$x^2g(x)$
110110	$1 + x + x^3 + x^4$	$g(x) + xg(x)$
011011	$x + x^2 + x^4 + x^5$	$xg(x) + x^2g(x)$
101101	$1 + x^2 + x^3 + x^5$	$g(x) + x^2g(x)$
111111	$1 + x + x^2 + x^3 + x^4 + x^5$	$(1 + x + x^2)g(x)$

Bilag 7 - Detaljeret process af erasure korrigerig

Given den modtagne besked $R(x)$, den transmitterede besked $T(x)$ og fejlen $E(x)$.

$$R(x) = T(x) + E(x) \quad (51)$$

Dekoderen skal finde korruptions polynomiet $E(x)$, så beskeden kan rettes:

$$T(x) = R(x) + E(x) \quad (52)$$

Da syndromet er perfekt dividerbar med generator polynomiet $g(x)$, hvilket betyder at hvis der ikke er sket en fejl så må:

$$\sum_{i=0}^{n-k-1} g(\alpha^i) = 0 \quad (53)$$

Syndrom vektoren er en vektor hvor hver element korresponderer til den en byte i kodeordet. Hvert element beregnes ved at evaluere $R(\alpha^i)$:

$$\begin{aligned} S_i &= R(\alpha^i) \\ &= T(\alpha^i) + E(\alpha^i) \\ &= E(\alpha^i), \quad \text{da } T(\alpha^i) = 0 \\ &= Y_1 X_1^i + Y_2 X_2^i + \dots + Y_v X_v^i \end{aligned} \quad (54)$$

Error locator polynomiet er defineret som:

$$\Lambda(x) = (1 + X_1 x)(1 + X_2 x) \dots (1 + X_v x) \quad (55)$$

Og rødderne kan findes ved Berlekamp–Massey algoritme. Til sidst bruges Forney algoritme til at beregne magnituden af hver koefficient, og korruptions vektor polynomiet $E(x)$, tilføjes til dataet for at korrigere fejlen, se (52).

Bilag 8 - Kildekode af hamming (7,4) codec

Kildekoden kan findes på github: <https://github.com/Sveske-Juice/hamming7-4-codec>. Kildekoden består af facade funktionerne som beskrevet i implementationen, deres sub-

rutiner (`hamming7-4.cpp`), og noget driver kode (`main.cpp`), som står for at tage input fra brugeren og kalde de rigtige facade funktioner `encode()` el. `decode()`.

Bilag 9 - Kildekode af RS codec

Den kommenterede kildekode for RS-codec kan findes på github: <https://github.com/Sveske-Juice/reed-solomon-codec>.

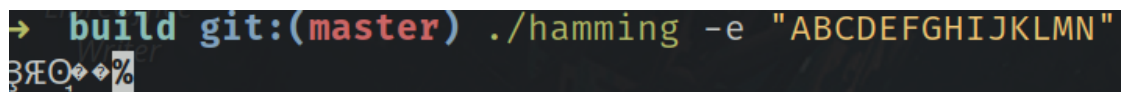
Bilag 10 - Test af hamming koden

Når programmet er kompileret, virker det som et *command line* program, der følger populære command line programmer fra GNU Core.

For at enkode, bruges `"-e"` flaget:

```
$ ./hamming -e "ABCDEFGHJKLMN"
```

Som printer:

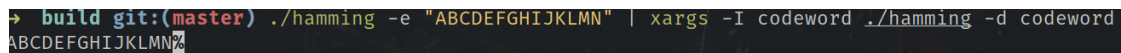


```
→ build git:(master) ./hamming -e "ABCDEFGHJKLMN"
3ÆQ♦♦%
```

Det er tydeligt at beskeden er enkodet, da den ikke kan blive repræsenteres som karaktere. For dekodning bruges `"-d"` flaget:

```
$ ./hamming -e "ABCDEFGHJKLMN" | xargs -I codeword ./hamming -d codeword
```

Da kodeordet indeholder null-karakterer og escape sekvenser, kan jeg ikke kopiere direkte, og piper derfor kodeordet til `xargs` kommandoen, som indsætter kodeordet efter `"-d"`. Det printe



```
→ build git:(master) ./hamming -e "ABCDEFGHJKLMN" | xargs -I codeword ./hamming -d codeword
ABCDEFGHJKLMN%
```

Det observeres at den oprindelige besked er blevet printet. Nu introduceres et bit-flip med **Bilag 11 - Bit flip program**:

```
$ ./hamming -e "ABCDEFGHJKLMN" | xargs -I codeword ./bitflipper codeword 3
```

Den flipper det 3. bit i hele kodeordet hvilket gør at den enkodet streng nu er korruperet:

```
→ build git:(master) ./hamming -e "ABCDEFGH IJKLMN" | xargs -I codeword ./bitflipper codeword 3
```

Nu bliver den korruperede kodeord pipet til dekoderen:

```
$ ./hamming -e "ABCDEFGH IJKLMN" |
  xargs -I codeword ./bitflipper codeword 3 |
  xargs -I corrupted ./hamming -d corrupted
```

Og fejlen bliver korrigeret, og den originale besked returneres!:

```
→ build git:(master) ./hamming -e "ABCDEFGH IJKLMN" | xargs -I codeword ./bitflipper codeword 3 | xargs -I corrupted ./hamming -d corrupted
ABCDEFGH IJKLMN
```

Der er blevet udviklet en række unit tests som tester forskellige dele af codec'et, se [github](#).

Bilag 11 - Bit flip program

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 void flipBit(char *str, int bitPosition) {
6     int index = bitPosition / 8; // Calculate the byte index
7     int bitOffset = bitPosition % 8; // Calculate the bit offset within
   the byte
8
9     str[index] ^= (1 << bitOffset); // Flip the specified bit
10 }
11
12 int main(int argc, char *argv[]) {
13     if (argc < 2) {
14         printf("Usage: %s \"<my cool string>\" n1 n2 n3 ...\\n", argv[0]);
15         return 1;
16     }
17
18     // Extract the input string from the command line arguments
19     char *inputString = argv[1];
20
21     // Iterate through the provided bit positions and flip the
   corresponding bits
```

```
22     for (int i = 2; i < argc; i++) {
23         int bitPosition = atoi(argv[i]);
24
25         if (bitPosition < 0 || bitPosition >= strlen(inputString) * 8) {
26             printf("Invalid bit position: %d\n", bitPosition);
27             return 1;
28         }
29
30         flipBit(inputString, bitPosition);
31     }
32     printf("%s", inputString);
33     return 0;
34 }
```

Listing 6: Program til at simulere bit flips