McEliece, kodebaseret kryptografi og matrixprodukt-koder

Speciale

Kodningsteori

29. januar 2016

Camilla Lund Melander • **Ida Bentzen-Petersen**

Matematisk Institut

Fredrik Bajers Vej 7G • 9220 Aalborg Ø • Tlf. 99409940

Abstract

This project has been written under the subject coding theory and takes its focus on the McEliece cryptosystem. The main disadvantage of this cryptosystem is the large public key size and therefore we have researched the possibility of reducing it by using matrix-product codes. In the first chapter, we present the McEliece cryptosystem and also McEliece's original proposal to use Goppa codes for the cryptosystem. We also consider the equivalent cryptosystem developed by Nieddereiter and thus a comparison of these two cryptosystems is given. From here, we only focus on the McEliece cryptosystem due to the fact that these cryptosystems are equivalent.

Chapter two considers two kinds of attacks on the cryptosystem namely message attack and key attack. These attacks have been used against the McEliece cryptosystem with different families of codes. In this chapter it is shown through examples that breaking the McEliece cryptosystem is very expensive even for small parameters of the code. It is further noted that some types of codes are still secure meaning that they can resist known attacks. One of these types are irreducible binary Goppa codes.

We aim to reduce the public key size in the cryptosystem and one way of doing this is by looking at codes that have a quasi cyclic structure. Therefore the theory of these codes is presented in chapter three. Also in this chapter we introduce Gaborit's version of the McEliece cryptosystem. This version uses the quasi cyclic structure and a special way of generating the generator matrix to reduce the public key size in the McEliecce cryptosystem. Furthermore, he changes the key generating algorithm in such a way that the generator matrix is now only hidden by a permutation matrix. In Gaborit's version he uses subcodes of known BCH-codes, and by using this it has been shown that the system can be broken. Even though, the version has been broken, the main idea can be used to develop a new version of the McEliece cryptosystem using matrix-product codes.

In chapter four, we present exactly this type of codes. Among others it is shown, that the minimum distance of these codes can be calculated exactly under given conditions. Moreover, a decoding algorithm for matrix-product codes is given, when the codes are nested and the matrix is non-singular by columns. Another example of using this type of codes in the McEliece cryptosystem is provided.

A new version of the McEliece cryptosystem is presented in chapter five. This version uses matrix-product codes, and Gaborit's idea of finding a permutation in such a way, that the permutations only are within the small parts of the codeword. Furthermore the

new version uses the fact that chapter four showed that a generator matrix for a matrix-product code can be constructed from the small codes generator matrices and the matrix A. Knowing this it is now possible to reduce the public key size of the McEliece cryptosystem by only sending these relatively small generator matrices and matrix A in stead of a very large generator matrix for the entire matrix-product code.

When Gaborit's idea is used for the new version of the McEliece cryptosystem, one might think that the new version such as Gaborit's can be broken. It is suggested, that if irreducible binary nested Goppa codes are used it might not be possible to break the version, since this is one of the code types that are still secure against attacks. Furthermore, it is stated that for future work it might be interesting to look at subfield subcodes and use this family of codes for the new version. This idea is also proposed by Berger for Gaborit's version of the McEliece cryptosystem.

AALBORG UNIVERSITET
STUDENTERRAPPORT

Institut for Matematiske Fag

Matematik og Statistik

Fredrik Bajers Vej 7G

Tlf. 99409940

http://math.aau.dk

Titel:

McEliece, kodebaseret kryptografi og matrixprodukt-koder

Tema:

Kodningsteori

Projektperiode:

Efterårssemesteret 2015

Projektgruppe:

G2-106cd

Deltagere:

Camilla Lund Melander

Ida Bentzen-Petersen

Vejleder: Diego Ruano

Oplagstal: 5

Sidetal: 100 (160 inklusiv appendiks)

Afsluttet den: 29. januar 2016

Synopsis:

Dette projekt omhandler kodningsteori med fokus på McEliece kryptosystemet. Først behandles opbygningen af McEliece kryptosystemet samt Niederreiter kryptosystemet, og i denforbindelse gennemgås teorien om Goppa-koder. En væsentlig del af projekt er sikkerheden af McEliece kryptosystemet, og i den forbindelse arbejdes der med to typer af angreb, meddelelsesangreb og nøgleangreb. Herefter præsenteres teorien om quasicykliske koder samt Gaborits version, der netop benytter disse koder struktur til at reducere den offentlige nøglestørrelse. Idéen bag denne reduktion benttes i projektet til at udvikle en ny version af McEliece kryptosystemet, der benytter matrixprodukt-koder.

Rapportens indhold er frit tilgængeligt, men offentliggørelse (med kildeangivelse) må kun ske efter aftale med forfatterne.

Forord

Dette projekt er udarbejdet som speciale i matematik ved Institut for Matematiske Fag ved Aalborg Universitet, af Camilla Lund Melander og Ida Bentzen-Petersen, i perioden 1. september 2015 til 29. januar 2016.

En stor tak til vores vejleder Diego Ruano, der har været til stor hjælp i forbindelse med projektet.

Projektet er udarbejdet under emnet „*McEliece, kodebaseret kryptografi og matrixprodukt-koder*“, der er valgt ud fra det overordnede tema „*Kodningsteori*“.

Læsevejledning

Læseren gøres opmærksom på, at det antages, at læseren har en grundlæggende viden indenfor kodningsteori, herunder viden omkring lineære blokkoder og i særdeleshed Reed-Solomon-koder. Vektorer i projektet er rækkevektorer, medmindre andet er angivet, og disse noteres med fed skrift. Læseren gøres opmærksom på at alle koder i dette projekt er lineære og en kode med parametrene, længde n, dimension k og minimumsafstand d noteres som en $[n, k, d]$-kode. I de tilfælde, hvor d ikke kendes noteres blot en $[n, k]$-kode. Reed-Solomon-koder noteres ved $RS_q[n, k]$, hvor q er alfabetet. De naturlige tal betragtes som mængden $\mathbb{N} = \{1, 2, \ldots\}$. Litteraturhenvisninger er anført ved (Forfatter, årstal), og hvis et kapitel eller afsnit bygger på samme kilde, angives det i starten. Igennem projektet nummereres eksempler, definitioner, sætninger og lemmaer samlet, hvor de første to tal angiver henholdsvis kapitel og afsnit, mens det sidste tal angiver den kronologiske orden i afsnittet. Derudover vil beviser blive afsluttet med ■, og eksempler med ◀. Figurer, tabeller og algoritmer nummereres separat.

Indholdsfortegnelse

Introduktion

Dette projekt omhandler McEliece kryptosystem, der er en form for kryptografi. Ordet kryptografi er en sammensætning af de to græske ord *kryptos* og *grafein*, der betyder henholdsvis „skjult“ og „at skrive“. Et af de mest simple kryptosystemer er kryptering, hvor et bogstav byttes med et andet fra samme alfabet. Denne form for kryptering er dog usikker, da nogle bogstaver forekommer oftere end andre, og derved kan statistik benyttes til forholdsvis simpel dekryptering.

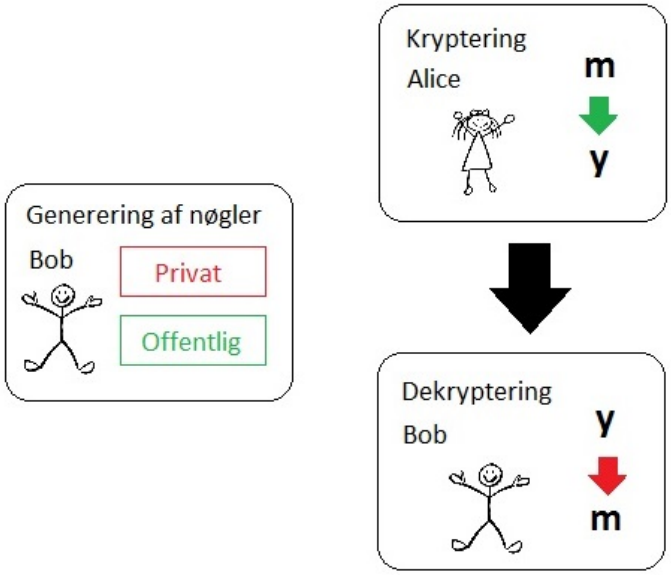
Gennem tiden er kryptering blevet udviklet i takt med, at de forskellige kryptosystemer kunne brydes. Den grundlæggende viden, der præsenteres i introduktionen bygger på (Bauer, 2013), (Klein, 2014) og (Márquez-Corbella m.fl., 2015). Nogle af de faktorer, der har haft stor indflydelse på udviklingen er opfindelsen og udviklingen af computeren samt krigsførsel, hvor især Anden Verdenskrig har spillet en stor rolle i form af Enigma. Tyskerne brugte Enigma til at indkode meddelelser før de blev sendt, og på denne måde kunne de allierede ikke umiddelbart læse meddelelserne. Alan Turing opfandt en maskine, som de allierede kunne benytte til at bryde Enigma-krypteringen, da denne maskine på kort tid kunne gennemløbe de forskellige mulige indstillinger for Enigma. Dette er begyndelsen på det, der i dag kendes som en computer.

Allan Turing mødte i forbindelse med sine studier amerikaneren Claude Shannon. De arbejdede begge på kryptering af stemmer til brug ved kommunikation mellem den amerikanske præsident Franklin D. Roosevelt og den brittiske premierminister Winston Churchill under krigstiden. Udover dette er Shannon kendt for at udvikle en regel for hvor lang en ciffertekst, der skal sendes for at en entydig løsning garanteres. Sagt på en anden måde er det en regel der siger, hvor mange bits, der skal sendes for at den oprindelige meddelelse entydigt kan opnås. I daglig tale siges Shannon at være grundlæggeren af informationsteori.

I perioden efter Anden Verdenskrig arbejdes med symmetrisk kryptografi hvor afsender og modtager benytter samme hemmelige nøgle til både indkodning og dekodning. Denne form for kryptografi er upraktisk, da det er nødvendigt at modtageren og afsenderen mødes for at udveksle nøglen, eller sammen skal blive enige om hvordan nøglen skal genereres. Symmetrisk kryptografi kaldes også hemmmelig nøgle kryptografi, og David Kahn udgav i 1967 bogen *The Codebreakers - The Story of Secret Writing*, der netop omhandler dette. Denne bog er blevet inspirationskilden for flere udviklinger af kryptosystemer.

I 1976 introducerede de to amerikanere Whitfield Diffie og Martin E. Hellman konceptet om offentlig nøgle kryptosystemer, netop med inspiration fra denne bog.

Konceptet bag disse er, at hvis Alice ønsker at sende en meddelelse til Bob, så foregår det ved, at Bob genererer to nøgler. Han genererer én privat og én offentlig nøgle. Den offentlige nøgle er frit tilgængelig, og derfor behøver Alice og Bob ikke at udveksle nøgler på forhånd. Det er den offentlige nøgle, som Alice benytter til at indkode meddelelsen. Den private nøgle benytter Bob til at dekode meddelelsen fra Alice. De to nøgler er genereret således, at det er beregningsteknisk umuligt at opnå den private nøgle ud fra den offentlige nøgle i polynomiel tid. Det vil sige, at der er tale om en *trap door*, en funktion, der kun går den ene vej, hvilket vil sige at den private nøgle ikke kan opnås ud fra den offentlige nøgle, mens det omvendte er muligt.



Figur 1. Bob genererer en privat og en offentlig nøgle. Alice ønsker at sende en meddelelse til Bob og benytter den offentlige nøgle til at indkode meddelelsen $\mathbf{m}$ til cifferteksten $\mathbf{y}$. Bob benytter den private nøgle til at dekode cifferteksten $\mathbf{y}$ tilbage til meddelelsen $\mathbf{m}$.

Der er flere forskellige former for kryptografi og et de mest kendte kryptosystemer er *RSA*. RSA er et offentligt nøgle system, og navnet er en forkortelse for efternavnene på de tre matematikere, Ron Rivest, Adi Shamir og Len Adleman, der opfandt systemet i 1977. Konceptet bag RSA-kryptering er, at vælge to store primtal p og q og ud fra disse danne $n = pq$, der sammen med en hjælpeværdi udgør den offentlige nøgle. Alle der kender den offentlige nøgle kan altså indkode ved hjælp af RSA, men det er kun de personer, der kender primtallene, som kan dekode, når primtallene er tilpas store. Ulemperne ved RSA er, at algoritmerne til ind- og dekodning er langsomme, da de bygger på faktorisering. Hvis det er muligt at faktorisere i polynomiel tid, hvilket netop er tilfældet, når der tales om kvantecomputere, vil RSA være usikkert.

Kvantecomputere blev omtalt første gang i slutningen af 1960'erne. Det er allerede i dag lykkedes at fremstille kvantecomputere, men de er meget små, og de benyttes derfor ikke i praksis. Fremstillingen af kvantecomputere der kan benyttes i praksis, er dog noget af det, der forskes meget i. Når det lykkes at fremstille en større kvantecomputer, vil det

som nævnt tidligere ikke være sikkert at benytte RSA til udveksling af information. Robert J. McEliece har dog allerede i 1978 konstrueret et kryptosystem, McEliece kryptosystemet, der er sikkert overfor kvantecomputere, når kodetype og parametre vælges smart. McEliece kryptosystemet er et offentligt nøgle kryptosystem, hvor den offentlige nøgle består af en generatormatrix samt en fejlvægt t. I sin oprindelige fremstilling benyttede McEliece binære Goppa-koder, og mere specifikt Goppa-koden med en længde på 1024 bits, dimension på 524 samt minimumsafstand på 101, og dermed en tilladt fejlvægt på 50.

Når McEliece kryptosystemet er sikkert overfor kvantecomputere, hvorfor benyttes dette så ikke allerede i dag til kryptering? Kryptosystemets store ulempe, og dermed den primære grund til at McEliece kryptosystemet ikke har vundet indpas i den praktiske brug af kryptografi er, at den offentlige nøglestørrelse er meget stor. Dermed er det beregningsmæssigt dyrt at benytte McEliece kryptosystemet fremfor RSA, og så længe kvantecomputere ikke er udviklet, er der ingen grund til denne ekstra arbejdsbyrde. Da der er udsigt til fremstilling af en kvantecomputer er det dog i høj grad relevant at arbejde med McEliece kryptosystemet, og forbedringer af dette. Dermed vil det stadig være muligt at sende meddelelser sikkert, selv når en kvantecomputer benyttes i et forsøg på at bryde krypteringen.

I et forsøg på at forbedre den store offentlige nøgle i McEliece kryptosystemet videreudviklede Harald Niederreiter McEliece kryptosystemet. Ideen bag Niederreiter kryptosystemet er den samme som for McEliece kryptosystemet, hvor det dog i stedet for en generatormatrix er en paritetstjekmatrix, der benyttes. Fordelen ved Niederreiter kryptosystemet er netop den mindre offentlige nøgle, men en ulempe er at algoritmerne, der skal benyttes til indkodning og dekodning er langsommere end dem for McEliece kryptosystemet.

Det typiske koncept bag forbedringen af McEliece kryptosystemet er dog ikke som Niecerreiter at ændre systemet, men derimod at forsøge med andre kodetyper, og dermed nedsætte størrelsen på den offentlige nøgle. Mange af disse „nye“ kodetyper til McEliece er dog blevet brudt. Der findes to overordnede typer af angreb på McEliece kryptosystemet, *meddelelsesangreb* og *nøgleangreb*. Meddelelsesangreb prøver direkte at finde den oprindelige meddelelse ud fra cifferteksten, hvor nøgleangreb istedet finder frem til kodens struktur, og derefter benytter en kendt dekodningsalgoritme til at opnå meddelelsen. Der er igennem tiden foreslået forskellige typer af koder, til McEliece og Niederreiter kryptosystemerne i forsøget på, at mindske størrelsen på den offentlige nøgle. Mange af disse forsøg er dog senere blevet brudt ved hjælp af et af ovenstående angreb.

I 1986 foreslår Niederreiter Generaliserede Reed-Solomon-koder med, længde 256, dimension 128 samt mininmumsafstand 129 til McEliece kryptosystemet (Niederreiter, 1986). Denne form for koder er dog brudt af V.M. Sidelnikov og O.V. Shestakov i 1992, da de formåede at opnå den private nøgle ved et nøgleangreb. Dette angreb er ikke kun effektivt for en generaliseret Reed-Solomon-kode med disse parametre, men virker generelt når der benyttes generaliserede Reed-Solomon-koder til både Niederreiter samt McEliece kryptosystemet (Sidelnikov m.fl., 1992). I samme artikel foreslår Niederreiter også konkatenerede koder til McEliece kryptosystemet, dette brydes dog i 1998 af Nicolas

Sendrier ved brug af nøgleangreb (Sendrier, 1998).

Reed-Muller koder bliver i 1994 foreslået til McEliece kryptosystemet af V.M. Sidelnikov, for en kode med længde 1024, dimension 176 samt minimumsafstand 128. Dette foreslag er dog også blevet brudt i 2007 af L. Minder og A. Shokrollahi samt i 2013 I. V. Chizhov og M. A. Borodin ved hjælp af nøgleangreb. I 2013 viser I. V. Chizhov og M. A. Borodin at McEliece kryptosystemet kan brydes af en almindelig computer på bare 7 timer, når længden af koden er på hele 65536 bits (Chizhov m.fl., 2013).

Herefter bliver algebraiske geometrikoder i 1996 benyttet til McEliece kryptosystemet af H. Janwa og O. Moreno (Janwa m.fl., 1996). Denne type af koder bliver dog også brudt ved hjælp af nøgleangreb af C. Faure og L. Minder i 2008. Dette gøres ved en videreudvikling af algoritmen foreslået af V.M. Sidelnikov og O.V. Shestakov i 1992 (Faure m.fl., 2008). I 2014 udføres at angreb af A. Couvreur, I. Márquez-Corbella samt R. Pellikaan, der hverken falder ind under kategorien meddelelsesangreb eller nøgleangreb, som bryder McEliece kryptosystemet med algebraiske geometrikoder (Couvreur m.fl., 2014).

Delkoder af generaliserede Reed-Solomon-koder er også foreslået som en kodetype, der kan være sikre overfor angreb i Niederreiter kryptosystemet, dette gøres i 2005 af T. Berger og P. Loidreau (Berger m.fl., 2005). Allerede i 2010 bliver dette dog brudt af C. Wieschebrink for koder med en lille dimension, og da de to kryptosystemer, Niederreiter og McEliece, kan ses som ækvivalente er McEliece kryptosystemet med delkoder af generaliserede Reed-Solomon-koder også brudt. Dette gøres også ved et nøgleangreb, og det bemærkes, at irreducible binære Goppa-koder, der er alternerende koder, er delkoder af generaliserede Reed-Solomon-koder, stadig er sikre (Wieschebrink, 2010). Der er dermed stadig nogle kodetyper, der kan modstå nuværende kendte angeb. Dette er altså Goppa-koder, subfield subkoder af algebraiske geometrikoder samt alternerende koder.

Fokuspunktet i dette projekt er at undersøge, om den offentlige nøglestørrelse kan reduceres ved at benytte matrixprodukt-koder til McEliece kryptosystemet. Dette gøres ved først at betragte McEliece kryptosystemet, herunder den generelle tankegang, algoritmer samt fordele og ulemper ved dette. I den forbindelse præsenteres også det ækvivalente kryptosystem, Niederreiter kryptosystemet, og en sammenligning af algoritmerne for disse udarbejdes. For at få et indblik i McElieces oprindelige forslag præsenteres kodetypen Goppa-koder, der som tidligere nævnt stadig ikke er brudt for McEliece kryptosystemet. Som en udvidelse heraf er det i dette projekt valgt at lave et kapitel omhandlende de to typer af angreb, meddelelsesangreb samt nøgleangreb.

Ulempen ved McEliece kryptosystemet er den store offentlige nøgle, og det undersøges, om denne kan formindske ved at benytte en anden type koder. Derfor ses i dette projekt på henholdsvis cykliske og quasicykliske koder, der begge kan dannes specielt så den offentlige nøglestørrelse kan reduceres betydeligt. Herefter ses på matrixprodukt-koder, hvor parametrene udledes, og en dekodningsalgoritme præsenteres for det tilfælde, hvor delkoderne er indlejrede. Til sidst i projektet undersøges, om matrixprodukt-koder kan benyttes til McEliece kryptosystemet.

En af udfordringerne i projektet har været at finde information omkring de forskellige dele, der skal benyttes for at belyse McEliece kryptosystemet og dets egenskaber.

Litteraturen er derfor primært artikler, der er skrevet af forskellige teoretikere indenfor henholdsvis den matematiske og den datalogiske verden. En styrke ved projektet er netop at teorien er sammensat fra mange forskellige artikler, der har forskellige indgangsvinkler, fokuspunkter og notation.

Udover overvejelserne omkring hvilken teori, der skal benyttes er en stor del af projektet også forklarende eksempler, hvor beregningerne er lavet ved implementering i SAGE. Gennemgående arbejdes i eksemplerne med Reed-Solomon-koder, der på trods af at disse ikke er sikre overfor angreb illustrerer brugen af de forskellige algoritmer. Denne type koder er valgt, da de er MDS samt at der findes en kendt dekodningsalgoritme, der er forholdsvis nem at implementere.

Projektet er opbygget således, at det første kapitel behandler McEliece kryptosystemet, og herunder opbygningen og egenskaberne ved dette. Derfor bliver der i kapitlet også gennemgemgået teori om Goppa-koder, der er McElieces oprindeige foreslag til kodetype. Der gives desuden algoritmer til nøglegenerering, ind- og dekodning ved McEliece kryptosystemet. Derudover præsenteres Niederreiter kryptosystemet, der er ækvivalent med McEliece kryptosystemet, og en sammenligning af disse præsenteres. Da de to kryptosystemer er ækvivalente ses der efterfølgende kun på McEliece kryptosystemet, da teorien kan overføres til Niederreiter kryptosystemet.

Kapitel to omhandler sikkerheden af McEliece kryptosystemet, og omdrejningspunktet er behandlingen af de to typer af angreb, meddelelsesangreb og nøgleangreb. Til meddelelsesangreb benyttes metoden infromation set dekodning, og en algoritme til et sådant angreb præsenteres. Der gives et eksempel på et meddelelsesangreb med netop denne algoritme, og det vises at kompleksiteten for selv meget små parametre er voldsom. I afsnittet Nøgleangreb præsenteres the support splitting algorithm og teorien, der skal benyttes til denne. Der gives ikke direkte et eksempel på denne algoritme, men der gives en beskrivelse af fremgangsmåden ved brug af Goppa-koder.

Når det ønskes at reducere den offentlige nøglestørrelse er en idé at betragte quasicykliske koder og deres struktur. Derfor er det netop disse koder, der arbejdes med i kapitel tre. Kapitlet starter med en præsentation af cykliske koder, og teorien om disse udvides og benyttes til definition af samt egenskaber for quasicykliske koder. Herefter bliver en udgave af McEliece kryptosystemet, hvor den offentlige nøglestørrelse reduceres, beskrevet. Denne udgave er udviklet af Philippe Gaborit og bygger på den quasicykliske struktur. Tankegangen bag denne version er netop, hvad der ligger til grund for den nye version af McEliece kryptosystemet, der præsenteres i kapitel fem, hvor der benyttes matrixprodukt-koder. Yderligere gennemgås Gröbnerbasisrepræsentationen af quasicykliske koder, og en metode til at bestemme dimensionen gives.

I den nye version af McEliece kryptosystemet benyttes matrixprodukt-koder. Teorien om denne type koder gives i kapitel fire. Der vises blandt andet en grænse for minimumsafstanden samt hvad dimensionen for en sådan kode er. Desuden præsenteres begrædet indlejrede koder samt hvad der menes med at en matirx er ikke-singulær ved søjler. Når disse to begræber benyttes kan minimumsafstanden beregnes eksakt. Derudover benyttes de så en dekodningsalgoritme til matrixprodukt-koder kan findes. Når

matrixprodukt-koder på denne form benyttes til McEliece kryptosystemet er den offentlige nøglestørrelse stor, og derfor forsøges i kapitel fem at finde en metode til at reducere denne.

Kapitel fem bygger på den i projektet gennemgåede teori. Der arbejdes med idéen fra McEliece kryptosystemet, der er præsenteret i kapitel et, samt Gaborits idé, som er beskrevet i kapitel tre. Til denne version udarbejdes ligeledes algoritmer til nøglegenerering, ind- og dekodning. Disse algoritmer er blandt andet udviklet ud fra de egenskaber for matrixprodukt-koder, der præsenteres i kapitel fire, og i særdeleshed opbygningen af generatormatricen. I dette kapitel gøres der også oovervejelser omkring sikkerheden af den nye version, hvilket gøres ved at lave en sammenkobling mellem sikkerheden af Gaborits version og den nye version.

KAPITEL 1
McEliece kryptosystemet

Dette kapitel bygger på (Menezes m.fl., 1997), (Peters, 2011), (Berger m.fl., 2009), (Bernstein, 2009) and (Márquez-Corbella m.fl., 2015).

I 1978 fremlagde den amerikanske matematiker Robert James McEliece det første offentlig nøgle kryptosystem, der er baseret på lineære fejkorrigerende koder (McEliece, 1978). Princippet i McEliece kryptosystemet er at vælge en t-korrigerende lineær kode med længde n, og så skjule koden som en tilfældig lineær kode. Oprindeligt arbejdede McEliece med binære Goppa-koder, og senere har flere forskellige matematikere undersøgt, hvilke andre koder, der kan benyttes.

Der er forskellige fordele og ulemper ved McEliece kryptosystemet. En af fordelene ved kryptosystemet er, at det er nemt både at indkode og dekode, når den private nøgle kendes. Der er én stor ulempe ved McEliece kryptosystemet, og det er størrelsen på den offentlige nøgle. Det er efter McElieces udgivelse blevet undersøgt, om det er muligt at benytte hans kryptosystem til andre koder, hvormed det er muligt at have en offentlig nøgle, der er mindre. McEliece arbejdede som nævnt i introduktionen med Goppa-koden, der har parametrene $n = 1024$, $k = 524$ og $t = 50$. Bemærk, at hvis matricen i den offentlige nøgle er på systematisk form, så bliver størrelsen reduceret fra kn bits til $k(n-k)$ bits, da det blot er nødvendigt at betragte den ikke-trivielle del af matricen. McEliece arbejdede altså med en stor offentlig nøgle på $k(n-k) = 524(1024-524) = 262.000$ bits.

Gennem tiden er forskellige typer af koder forsøgt til McEliece kryptosystemet, for at reducere den offentlige nøglestørrelse. Disse koder, der blandt andet er Generaliserede Reed-Solomon-koder og Reed-Muller-koder, har dog som tidligere nævnt vist sig ikke at være sikre, og er alle blevet brudt, mens McElieces oprindelige forslag med Goppa-koder stadig holder.

1.1 Teori om McEliece kryptosystemet

For at benytte McEliece kryptosystemet betragtes en generatormatrix G for en lineær kode over $\mathbb{F}_q$ som en afbildning $\mathbb{F}_q^k \to \mathbb{F}_q^n$, der sender en meddelelse $\mathbf{m} \in \mathbb{F}_q^k$ til et element i $\mathbb{F}_q^n$. Det er vigtigt, at der eksisterer en kendt dekodningsalgoritme for koden, da denne skal være en del af den private nøgle, og benyttes af Bob til at dekode ordet han modtager til kodeordet, og derved den meddelelse, der er sendt.

Når der genereres nøgler arbejdes med *work factor*, der er en form for tidskompleksitet, som fortæller, hvor hurtigt det er muligt at bryde krypteringen. Når parametrene n, k

og t vælges, gøres det således, at work factoren er større end eller lig 2^κ, for et valgt κ. Det ses, at work factoren vokser eksponentielt, dermed bliver den meget større, når κ vælges bare en lille smule større. Det vælges altså hvor svært, det skal være at bryde krypteringen. Work factor noteres med $W_{q,n,d,t}$, og betegner kompleksiteten for den bedst kendte dekodningsalgoritme, der dekoder en t-korrigende lineær kode med længde n og dimension k over $\mathbb{F}_q$. For at danne den offentlige samt den private nøgle benyttes følgende algoritme.

Algoritme 1 Nøglegenerering i McEliece kryptosystemet

1: Vælg parametrene n, k og t således at $W_{q,n,d,t} \geq 2^\kappa$.
2: Bestem en tilfældig generatormatrix G for en $[n, k, 2t + 1]$ kode C.
3: Bestem en tilfældig $n \times n$ permutationsmatrix P.
4: Bestem en tilfældig $k \times k$ invertibel matrix S.
5: Beregn $\hat{G} = SGP$.
6: **return** Offentlig nøgle $(\hat{G}, t)$ og privat nøgle $(S, G, P,$ dekodningsalgoritme for $C)$.

I et McEliece kryptosystem består den private nøgle af en dekodningsalgoritme til den oprindelige kode, der er en t-korrigerende lineær kode over $\mathbb{F}_q$, C, med længde n og dimension k. Derudover består den af to tilfældigt genererede matricer, en $n \times n$ permutations matrix P og en invertibel $k \times k$ matrix S. Den offentlige nøgle er en beskrivelse af den skjulte kode, altså $k \times n$-matricen $\hat{G} = SGP$ samt vægten t, der garanteret kan rettes.

Der gives nu et eksempel på hvordan nøglegenereringen i McEliece kryptosystemet foregår. I eksemplet benyttes Reed-Solomon-koder. Det bemærkes dog, at det i 1992 af V. M. Sidelnikov og O. V. Shestakov er vist at McEliece kryptosystemet kan brydes, når det er Generaliserede Reed-Solomon-koder, der benyttes. Der er derfor tale om et teoretisk eksempel, og ikke et, der vil blive benyttet i praksis, hvilket også skyldes at parametrene er meget små, og derfor ikke sikre.

1.1.1 Eksempel:

I dette eksempel arbejdes med en Reed-Solomon-kode over $\mathbb{F}_{13}$. I Algoritme 1 er 1. punkt at vælge parametrene til koden. Disse vælges til $n = 12$, $k = 3$ og $t = 4$.

punkt 2 i algoritmen er at bestemme en tilfældig generatormatrix G for koden, hvilket er en $[12, 3]$-kode. Det vides, at en Reed-Solomon-kode har en generatormatrix på formen

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \cdots & x_n^{k-1} \end{bmatrix},$$

hvor $x_i = \beta^{i-1}$ for $i = 1, \ldots, n$ og β er primitivt element i $\mathbb{F}_{13}$. Et primitivt element i $\mathbb{F}_{13}$

er 2, derfor bliver x_i'erne

$$x_1 = 2^0 = 1 \qquad x_2 = 2^1 = 2 \qquad x_3 = 2^2 = 4$$
$$x_4 = 2^3 = 8 \qquad x_5 = 2^4 = 3 \qquad x_6 = 2^5 = 6$$
$$x_7 = 2^6 = 12 \qquad x_8 = 2^7 = 11 \qquad x_9 = 2^8 = 9$$
$$x_{10} = 2^9 = 5 \qquad x_{11} = 2^{10} = 10 \qquad x_{12} = 2^{11} = 7$$

Dermed bliver generatormatricen

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 3 & 6 & 12 & 11 & 9 & 5 & 10 & 7 \\ 1 & 4 & 3 & 12 & 9 & 10 & 1 & 4 & 3 & 12 & 9 & 10 \end{bmatrix}.$$

Udregningerne er lavet i SAGE og kan ses i appendiks B.1. Punkt 3 i Algoritme 1 er nu at vælge en tilfældig 12×12 permutationsmatrix

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

og punkt 4 er at vælge en tilfældig 3×3 invertibel matrix

$$S = \begin{bmatrix} 4 & 0 & 0 \\ 5 & 11 & 9 \\ 2 & 7 & 0 \end{bmatrix}.$$

Nu skal Bob beregne $\hat{G}$, hvilket er punkt 5 i algoritmen.

$$\hat{G} = SGP = \begin{bmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 6 & 11 & 3 & 12 & 2 & 3 & 1 & 6 & 1 & 11 & 5 & 12 \\ 1 & 3 & 12 & 9 & 10 & 8 & 7 & 6 & 0 & 4 & 5 & 11 \end{bmatrix}$$

Bob har nu dannet en generatormatrix, $\hat{G}$, der skal sendes skal den offentlige nøgle. Det er dog, som tidligere nævnt, muligt at reducere denne ved at sætte matricen på reduceret række-echelonform på følgende måde,

$$\hat{G} = \begin{bmatrix} 1 & 0 & 0 & 11 & 10 & 5 & 5 & 11 & 4 & 2 & 10 & 2 \\ 0 & 1 & 0 & 10 & 1 & 3 & 6 & 6 & 8 & 10 & 3 & 2 \\ 0 & 0 & 1 & 6 & 3 & 6 & 3 & 10 & 2 & 2 & 1 & 10 \end{bmatrix}$$

og det er dermed kun nødvendigt for Bob at sende en 3×9 matrix som den offentlige nøgle fremfor en 3×12 matrix. Udregningerne kan ses i appendiks B.1. I punkt 6 i algoritmen offentliggør Bob $\hat{G}$ og $t = 4$, men beholder resten af informationerne selv. ◀

Der er nu givet et eksempel på hvordan nøglegenereringen foregår i McEliece kryptosystemet. Det er med den offentlige nøgle Bob har genereret muligt for Alice at indkode en meddelelse og sende denne til Bob. Når Alice ønsker at sende meddelelsen til Bob, benytter hun følgende algoritme til at indkode.

Algoritme 2 Indkodning i McEliece kryptosystemet

Input: En meddelelse $\mathbf{m} \in \mathbb{F}_q^k$, den offentlige nøgle, $\hat{G}$ og parameteren t.
Output: En cifferskrift $\mathbf{y} \in \mathbb{F}_q^n$.

1: Beregn $\mathbf{m}\hat{G}$.
2: Bestem en tilfældig vektor $\mathbf{e}$ med længde n og vægt w og beregn $\mathbf{y} = \mathbf{m}\hat{G} + \mathbf{e}$.
3: **return** Cifferskriften $\mathbf{y}$.

For at indkode en meddelelse ved McEliece kryptosystemet kræves at meddelelsen er dannet over $\mathbb{F}_q$ og har længden k. Desuden kræves at den offentlige nøgle, matricen $\hat{G}$ og samt parameteren t kendes. Metoden er, at meddelelsen indkodes til et kodeord ved hjælp af matricen $\hat{G}$. Dernæst adderes en fejlvektor med vægt w til kodeordet. Det er vigtigt, at den fejlvektor der tilføjes har denne vægt, da det vides, at koden er t-korrigerende og $w \leq t$. Dermed er det garanteret, at Bob kan dekode, da han kender kodens type og dermed en effektiv dekodningsalgoritme. der gives nu et eksempel på algoritme 2.

1.1.2 Eksempel:

Hvis Alice ønsker, at sende meddelelsen $\mathbf{m} = \begin{bmatrix} 4 & 7 & 9 \end{bmatrix}$ til Bob, så benytter hun algoritme 2 og dermed den offentlige nøgle fra eksempel 1.1.1. Bemærk, at Alice i dette eksempel ikke har modtaget en offentlig nøgle hvor generatormatricen er på systematisk form. Punkt 1 i algoritme 2 er, at Alice skal beregne $\mathbf{m}\hat{G}$.

$$\mathbf{m}\hat{G} = \begin{bmatrix} 2 & 3 & 2 & 12 & 3 & 5 & 8 & 8 & 10 & 12 & 5 & 4 \end{bmatrix}.$$

Udregningerne er lavet i SAGE og kan ses i appendiks B.2. Herefter er punkt 2, at Alice skal vælge en tilfældig fejlvektor med vægt w. Alice vælger $w = 4$, og følgende fejlvektor

$$\mathbf{e} = \begin{bmatrix} 0 & 0 & 7 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 11 & 0 \end{bmatrix}.$$

Hun beregner nu

$$\mathbf{y} = \mathbf{m}\hat{G} + \mathbf{e} = \begin{bmatrix} 2 & 3 & 9 & 12 & 3 & 5 & 10 & 9 & 10 & 12 & 3 & 4 \end{bmatrix},$$

og sender i punkt 3 i algoritme 2 cifferteksten $\mathbf{y}$ til Bob. ◀

Når Bob modtager cifferteksten $\mathbf{y}$, skal han benytte den private nøgle til at dekode denne til meddelelsen Alice har sendt. Dette gør han ved at benytte følgende algoritme.

Algoritme 3 Dekodning i McEliece kryptosystemet

Input: En vektor $\mathbf{y} = \mathbf{m}\hat{G} + \mathbf{e} \in \mathbb{F}_q^n$ og den private nøgle (C, G, S, P).
Output: Meddelelsen $\mathbf{m}$.

1: Beregn $\hat{\mathbf{y}} = \mathbf{y}P^{-1} = \mathbf{m}SG + \mathbf{e}P^{-1}$, se ligning (1.1)
2: Benyt dekodningsalgoritmen for C til at finde $\mathbf{m}S$.
3: Brug lineær algebra til at finde $\mathbf{m}$, se ligning (1.2).
4: **return** Meddelelsen $\mathbf{m}$.

Når cifferskriften skal dekodes kræves det, at modtageren kender den private nøgle, (C, G, S, P). Dermed kan $\hat{G} = SGP$ beregnes. Metoden til dekodning er at udnytte, at matricerne P og S begge er invertibel. S er valgt som en invertible matrix, og det gælder, at enhver permutationsmatrix, og dermed også P, er invertibel med $P^{-1} = P^T$. Når den private nøgle kendes, kan P^{-1} og S^{-1} beregnes.

Første skridt i dekodningen er beregningen af $\hat{\mathbf{y}}$. Denne beregning foregår på følgende måde

$$\hat{\mathbf{y}} = \mathbf{y}P^{-1} = (\mathbf{m}\hat{G} + \mathbf{e})P^{-1} = (\mathbf{m}SGP + \mathbf{e})P^{-1} = \mathbf{m}SG + \mathbf{e}P^{-1}. \tag{1.1}$$

Vægten af $\mathbf{e}P^{-1}$ er w, hvilket er mindre end eller lig t. Dermed kan Bob benytte, at han kender kodens type samt en dekodningsalgoritme, for denne. På den måde opnår Bob $\hat{\mathbf{m}} = \mathbf{m}S$, og ved brug af S^{-1} kan han dekode $\hat{\mathbf{m}}$ til den oprindelige meddelelse

$$\mathbf{m} = \hat{\mathbf{m}}S^{-1}. \tag{1.2}$$

Der gives nu et eksempel på dekodning ved brug af algoritme 3

1.1.3 Eksempel:

Bob har modtaget cifferteksten

$$\mathbf{y} = \begin{bmatrix} 2 & 3 & 9 & 12 & 3 & 5 & 10 & 9 & 10 & 12 & 3 & 4 \end{bmatrix}$$

fra Alice, og han benytter algoritme 3 til at dekode. I 1. punkt i algoritmen beregner han P^{-1}, der svarer til P^T og dermed får han

$$\hat{\mathbf{y}} = \mathbf{y}P^{-1} = \begin{bmatrix} 12 & 3 & 12 & 9 & 3 & 3 & 5 & 2 & 10 & 4 & 10 & 9 \end{bmatrix}.$$

Da Bob ved, at koden er en Reed-Solomon-kode, kan han nu benytte en dekodningsalgoritme til at dekode $\hat{\mathbf{y}}$ til $\mathbf{m}S$, hvilket er punkt to i algoritmen. Han benytter algoritmen fra appendiks A.1. Først beregner Bob $\ell_0 = n - t - 1 = 12 - 4 - 1 = 7$ og $\ell_1 = t = 4$, og derefter løser han det homogene lineære ligningssystem bestående af 12 ligninger med 13

ubekendte,

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 12 & 12 & 12 & 12 \\ 1 & 2 & 4 & 8 & 3 & 6 & 12 & 11 & 3 & 6 & 12 & 11 & 9 \\ 1 & 4 & 3 & 12 & 9 & 10 & 1 & 4 & 12 & 9 & 10 & 1 & 4 \\ 1 & 8 & 12 & 5 & 1 & 8 & 12 & 5 & 9 & 7 & 4 & 6 & 9 \\ 1 & 3 & 9 & 1 & 3 & 9 & 1 & 3 & 3 & 9 & 1 & 3 & 9 \\ 1 & 6 & 10 & 8 & 9 & 2 & 12 & 7 & 3 & 5 & 4 & 11 & 1 \\ 1 & 12 & 1 & 12 & 1 & 12 & 1 & 12 & 5 & 8 & 5 & 8 & 5 \\ 1 & 11 & 4 & 5 & 3 & 7 & 12 & 2 & 2 & 9 & 8 & 10 & 6 \\ 1 & 9 & 3 & 1 & 9 & 3 & 1 & 9 & 10 & 12 & 4 & 10 & 12 \\ 1 & 5 & 12 & 8 & 1 & 5 & 12 & 8 & 4 & 7 & 9 & 6 & 4 \\ 1 & 10 & 9 & 12 & 3 & 4 & 1 & 10 & 10 & 9 & 12 & 3 & 4 \\ 1 & 7 & 10 & 5 & 9 & 11 & 12 & 6 & 9 & 11 & 12 & 6 & 3 \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ Q_{0,3} \\ Q_{0,4} \\ Q_{0,5} \\ Q_{0,6} \\ Q_{0,7} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ Q_{1,3} \\ Q_{1,4} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Han sætter matricen på reduceret-række-echelon-form

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 11 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 \end{bmatrix},$$

og vælger $Q_{1,4}$ som den frie variabel med værdien 1. Dermed opnår han

$$\begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ Q_{0,3} \\ Q_{0,4} \\ Q_{0,5} \\ Q_{0,6} \\ Q_{0,7} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ Q_{1,3} \\ Q_{1,4} \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 12 \\ 5 \\ 4 \\ 6 \\ 2 \\ 0 \\ 6 \\ 11 \\ 5 \\ 8 \\ 1 \end{bmatrix},$$

der giver polynomierne

$$Q_0(x) = 2 + 12x^2 + 5x^3 + 4x^4 + 6x^5 + 2x^6$$
$$Q_1(x) = 6 + 11x + 5x^2 + 8x^3 + x^4.$$

Dermed opnås

$$f(x) = -\frac{Q_0(x)}{Q_1(x)} = 4 + 10x + 11x^2,$$

der svarer til $\mathbf{m}S = \begin{bmatrix} 4 & 10 & 11 \end{bmatrix}$. Punkt tre i algoritme 3 er at finde meddelelsen $\mathbf{m}$, derfor udregner Bob

$$\mathbf{m} = \mathbf{m}SS^{-1} = \begin{bmatrix} 4 & 10 & 11 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 \\ 12 & 0 & 2 \\ 0 & 3 & 12 \end{bmatrix} = \begin{bmatrix} 4 & 7 & 9 \end{bmatrix}.$$

Det ses at meddelelsen, som Bob har dekodet til, netop er den meddelelse som Alice sendte. Alle beregninger fra eksemplet er lavet i SAGE og kan ses i appendiks B.3. ◀

Da McEliece udviklede sit kryptosystem arbejdede han med binære Goppa-koder, der som tidligere nævnt stadig er sikre. Derfor vil terorien om disse blive beskrevet i følgende afsnit.

1.2 Goppa-koder

Dette afsnit bygger på kapitel 12 paragraf 3 i (MacWilliams m.fl., 1977) samt forelæsning 1.8 i (Márquez-Corbella m.fl., 2015). Goppa-koder er en del af den gruppe af koder, der kaldes *alternerende koder*. Disse koder er subfield subkoder af generaliserede Reed-Solomon-koder, for definition af disse se appendiks A.4. Fordelen ved disse koder er, at det er muligt at arbejde med store koder over et lille legeme, altså at have et stort n i forhold til q.

For at kunne definere en Goppa-kode med længde n i $\mathbb{F}_q$ betragtes delmængden $L = \{\alpha_1, \ldots, \alpha_n\} \in \mathbb{F}_{q^m}^n$, hvor m er en valgt konstant og $\alpha_i \neq \alpha_j \ \forall i \neq j$. Derudover betragtes *Goppapolynomiet* $g(z)$, der er et monisk polynomium med koefficienter i $\mathbb{F}_{q^m}$. Det gælder, at $\deg(g) = r$ og at $g(\alpha_i) \neq 0 \ \forall \alpha_i \in L$. Det er desuden nødvendigt at indføre en funktion

$$R_{\mathbf{c}}(z) = \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} \tag{1.3}$$

for ethvert kodeord $\mathbf{c} = \begin{bmatrix} c_1 & c_2 & \ldots & c_n \end{bmatrix}$ med værdier i $\mathbb{F}_q$. Denne funktion benyttes til følgende definition af Goppa-koder.

1.2.1 Definition:

En Goppa-kode $\Gamma(L, g)$ er en lineær kode over $\mathbb{F}_q$, der består af alle vektorer $\mathbf{c} \in \mathbb{F}_q^n$, hvor

$$R_{\mathbf{c}}(z) \equiv 0 \mod g(z). \tag{1.4}$$

Ligning 1.4 svarer til, at $R_{\mathbf{c}}(z) = 0$ i polynomiumsringen $\mathbb{F}_{q^m}[z]/g(z)$. Bemærk, at når Goppapolynomiet $g(z)$ er irreducibelt, så siges Goppa-koden $\Gamma(L, g)$ at være en irreducibel Goppa-kode. Goppa-koden $\Gamma(L, g)$ er den alternerende kode $\mathcal{A}(\alpha, y)$, hvor $y_i = g(\alpha_i)^{-1}$, for definitionen af alternerende koder se appendiks A.4.

Længden af en Goppa-kode svarer til kardinaliteten af mængden L, og er altså $n = |L|$. For en Goppa-kode gælder desuden, at dimensionen opfylder uligheden $k \geq n - mr$, hvor $r = \deg(g)$. Minimumsafstanden har ligeledes en nedre grænse, og denne er givet ved $d \geq r + 1$. I det binære tilfælde gælder, at hvis Goppapolynomiet $g(z)$ ikke har nogle multiple rødder, og dermed også er irreducibelt, så opfylder minimumsafstanden en strengere ulighed $d \geq 2r + 1$. Dette er en fordel, da den nedre grænse for det antal fejl, der garanteret kan rettes er større. Der gives nu et eksempel på en binær Goppa-kode og dennes parametre.

1.2.2 Eksempel:

I dette eksempel arbejdes der over legemet $\mathbb{F}_{2^3}$. Som Goppapolynomium vælges

$$g(z) = z^2 + z + 1.$$

Der skal nu vælges en delmængde L af $\mathbb{F}_8$, sådan at $g(\alpha_i) \neq 0 \ \forall \alpha_i \in L$. Elementerne i mængden L kan dermed vælges som de elementer, der genereres af et primitivt element i $\mathbb{F}_8$. Vælges α som primitivt element i $\mathbb{F}_8$, kan der genereres otte forskellige elementer.

Betragt faktoriseringen $x^7 - 1 = (1 + x)(1 + x + x^3)(1 + x^2 + x^3)$, og vælg $1 + x + x^3$ som irreducibelt polynomium over $\mathbb{F}_8$ til at danne elementerne.

$$\begin{array}{lccccccl}
0 = & & & & & & & = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \\
1 = & 1 & & & & & & = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \\
\alpha = & & & \alpha & & & & = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \\
\alpha^2 = & & & & & \alpha^2 & & = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \\
\alpha^3 = & 1 & + & \alpha & & & & = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \\
\alpha^4 = & & & \alpha & + & \alpha^2 & & = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \\
\alpha^5 = & 1 & + & \alpha & + & \alpha^2 & & = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \\
\alpha^6 = & 1 & & & + & \alpha^2 & & = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}
\end{array}$$

Tabel 1. Mulige elementer i mængden L.

Elementerne er fundet ud fra følgende udregninger, og det bemærkes, at der kun

eksisterer otte forskellige, da $\alpha^7 = 1$.

$$\begin{aligned}
\alpha^3 &= \alpha + 1 \\
\alpha^4 &= \alpha \cdot \alpha^3 = \alpha(\alpha+1) = \alpha^2 + \alpha \\
\alpha^5 &= \alpha^2 \cdot \alpha^3 = \alpha^2(\alpha+1) = \alpha^3 + \alpha^2 = \alpha + 1 + \alpha^2 \\
\alpha^6 &= \alpha^3 \cdot \alpha^3 = (\alpha+1)(\alpha+1) = \alpha^2 + 1 \\
\alpha^7 &= \alpha^6 \cdot \alpha = (\alpha^2+1)\alpha = \alpha^3 + \alpha = \alpha + 1 + \alpha = 1
\end{aligned}$$

Det ses dermed, at mængden $\{0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}$ er elementerne i L, der altså i dette tilfælde er hele $\mathbb{F}_8$. Parametrene for Goppa-koden opfylder

$$\begin{aligned}
n &= |L| = 8 \\
k &\geq 8 - 3 \cdot 2 = 2 \\
d &\geq 2 \cdot 2 + 1 = 5.
\end{aligned}$$

◀

Når der arbejdes med Goppa-koder, kan paritetstjekmatricen findes ud fra ligning (1.4), først er det dog nødvendigt at betragte elementerne $z - \alpha_i$. Da α_i ikke er en rod i Goppapolynomiet vides, at $z - \alpha_i$ ikke dividerer $g(z)$. Når der yderligere arbejdes i en polynomiumsring modulo $g(z)$, må der eksistere en invers til $z - \alpha_i$. Det vises, at denne invers er på formen

$$(z - \alpha_i)^{-1} = -\frac{g(z) - g(\alpha_i)}{z - \alpha_i} g(\alpha_i)^{-1},$$

da følgende udregning gælder

$$\begin{aligned}
(z - \alpha_i)\left(-\frac{g(z) - g(\alpha_i)}{z - \alpha_i} g(\alpha_i)^{-1}\right) &= -(g(z) - g(\alpha_i))g(\alpha_i)^{-1} \\
&= -g(z)g(\alpha_i)^{-1} + 1 \\
&\equiv 1 \mod g(z).
\end{aligned}$$

Ligning 1.4 kan nu omskrives på følgende måde.

$$\begin{aligned}
\sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} &= \sum_{i=1}^{n} c_i (z - \alpha_i)^{-1} \\
&= \sum_{i=1}^{n} c_i \left(-\frac{g(z) - g(\alpha_i)}{z - \alpha_i} g(\alpha_i)^{-1}\right) \\
&= -\sum_{i=1}^{n} c_i \left(\frac{g(z) - g(\alpha_i)}{z - \alpha_i} g(\alpha_i)^{-1}\right) \qquad (1.5)
\end{aligned}$$

Et kodeord er i Goppa-koden, $\mathbf{c} \in \Gamma(L, g)$, hvis og kun hvis ligning (1.5) er lig nul. Dette svarer til følgende lighed, da fortegnet er irrelevant ved værdien nul.

$$\sum_{i=1}^{n} c_i \frac{g(z) - g(\alpha_i)}{z - \alpha_i} g(\alpha_i)^{-1} = 0. \qquad (1.6)$$

Nu kan venstresiden i ligning (1.6) betragtes som et polynomium, der er funktionen $R_{\mathbf{c}}(z)$ fra ligning 1.3. Lad Goppapolynomiet være givet ved

$$g(z) = \sum_{h=0}^{r} g_h z^h,$$

hvor $g_h \in \mathbb{F}_{q^m}$ og $g_r \neq 0$. Så kan brøken fra ligning (1.6) omskrives på følgende måde

$$\begin{aligned}\frac{g(z) - g(\alpha_i)}{z - \alpha_i} &= \frac{\left(\sum_{h=0}^{r} g_h z^h\right) - \left(\sum_{h=0}^{r} g_h \alpha_i^h\right)}{z - \alpha_i} \\ &= \frac{\sum_{h=0}^{r} g_h (z^h - \alpha_i^h)}{z - \alpha_i} \\ &= \frac{\sum_{h=1}^{r} g_h (z^h - \alpha_i^h)}{z - \alpha_i}.\end{aligned}$$

Sidste lighedstegn følger af, at leddet $g_0(z^0 - \alpha_i^0) = 0$. Det vides, at

$$z^h - \alpha_i^h = (z - \alpha_i)(z^{h-1} + z^{h-2}\alpha_i + z^{h-3}\alpha_i^2 + \cdots + z\alpha_i^{h-2} + \alpha_i^{h-1}).$$

Dermed opnås brøken

$$\frac{\sum_{h=1}^{r} g_h \left((z - \alpha_i)(z^{h-1} + z^{h-2}\alpha_i + z^{h-3}\alpha_i^2 + \cdots + z\alpha_i^{h-2} + \alpha_i^{h-1})\right)}{z - \alpha_i}$$

$$= \frac{(z - \alpha_i)\sum_{h=1}^{r} g_h (z^{h-1} + z^{h-2}\alpha_i + z^{h-3}\alpha_i^2 + \cdots + z\alpha_i^{h-2} + \alpha_i^{h-1})}{z - \alpha_i}$$

$$= \sum_{h=1}^{r} g_h (z^{h-1} + z^{h-2}\alpha_i + z^{h-3}\alpha_i^2 + \cdots + z\alpha_i^{h-2} + \alpha_i^{h-1})$$

$$= g_1 + g_2(z + \alpha_i) + \cdots + g_{r-1}(z^{r-2} + z^{r-3}\alpha_i + \cdots + \alpha_i^{r-2}) + g_r(z^{r-1} + z^{r-2}\alpha_i + \cdots + \alpha_i^{r-1}).$$

Dermed kan ligning (1.6) omskrives til

$$\begin{aligned}&\sum_{i=1}^{n} \mathbf{c}_i (g_1 + g_2(z + \alpha_i) + \cdots + g_{r-1}(z^{r-2} + z^{r-3}\alpha_i + \cdots + \alpha_i^{r-2}) \\ &\qquad + g_r(z^{r-1} + z^{r-2}\alpha_i + \cdots + \alpha_i^{r-1}))g(\alpha_i)^{-1} \\ &= \sum_{i=1}^{n} \mathbf{c}_i (g_1 g(\alpha_i)^{-1} + g_2 g(\alpha_i)^{-1}(z + \alpha_i) + \cdots + g_{r-1} g(\alpha_i)^{-1}(z^{r-2} + z^{r-3}\alpha_i + \cdots + \alpha_i^{r-2}) \\ &\qquad + g_r g(\alpha_i)^{-1}(z^{r-1} + z^{r-2}\alpha_i + \cdots + \alpha_i^{r-1})) \\ &= 0\end{aligned}$$

Da $\mathbf{c}$ er i Goppa-koden $\Gamma(L, g)$ hvis og kun hvis $H\mathbf{c}^T = \mathbf{0}$, ses at matricen H skal svare til, at første række i H er koefficienterne for z^{r-1}, anden række for z^{r-2} og så videre. Dermed opnås, at

$$H = \begin{bmatrix} g_r g(\alpha_1)^{-1} & \cdots & g_r g(\alpha_n)^{-1} \\ (g_{r-1} + \alpha_1 g_r) g(\alpha_1)^{-1} & \cdots & (g_{r-1} + \alpha_n g_r) g(\alpha_n)^{-1} \\ \vdots & \cdots & \vdots \\ (g_1 + \alpha_1 g_2 + \cdots + \alpha_1^{r-1} g_r) g(\alpha_1)^{-1} & \cdots & (g_1 + \alpha_n g_2 + \cdots + \alpha_n^{r-1} g_r) g(\alpha_n)^{-1} \end{bmatrix}$$

er en $(n-k)\times n$ paritetstjekmatrix. Matricen kan opskrives som et matrixprodukt $H = XYZ$, hvor

$$XY = \begin{bmatrix} g_r & \cdots & g_r \\ g_{r-1}+\alpha_1 g_r & \cdots & g_{r-1}+\alpha_n g_r \\ \vdots & \cdots & \vdots \\ g_1+\alpha_1 g_2+\cdots+\alpha_1^{r-1}g_r & \cdots & g_1+\alpha_n g_2+\cdots+\alpha_n^{r-1}g_r \end{bmatrix}$$

$$Z = \begin{bmatrix} g(\alpha_1)^{-1} & 0 & 0 & \cdots & 0 \\ 0 & g(\alpha_2)^{-1} & 0 & \cdots & 0 \\ 0 & 0 & g(\alpha_3)^{-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & g(\alpha_n)^{-1} \end{bmatrix}.$$

Paritetstjekmatricen kan yderligere skrives som produktet af tre matricer på følgende måde $H = XYZ$, hvor

$$X = \begin{bmatrix} g_r & 0 & 0 & \cdots & 0 \\ g_{r-1} & g_r & 0 & \cdots & 0 \\ g_{r-2} & g_{r-1} & g_r & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g_r \end{bmatrix}$$

$$Y = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_1^{r-1} & \alpha_2^{r-1} & \cdots & \alpha_n^{r-1} \end{bmatrix}$$

$$Z = \begin{bmatrix} g(\alpha_1)^{-1} & 0 & 0 & \cdots & 0 \\ 0 & g(\alpha_2)^{-1} & 0 & \cdots & 0 \\ 0 & 0 & g(\alpha_3)^{-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & g(\alpha_n)^{-1} \end{bmatrix}$$

Da matricen X er invertibel er også matricen $H' = YZ$ en mulig paritetstjekmatrix for koden $\Gamma(L,g)$. Dette ses, da et kodeord $\mathbf{c}$ opfylder $\mathbf{c}\in\Gamma(L,g) \Leftrightarrow H\mathbf{c}^T = \mathbf{0}$, og når X er invertibel gælder, at $X\mathbf{x} = \mathbf{0} \Rightarrow \mathbf{x} = \mathbf{0}$. Dermed opnås, at $H\mathbf{c}^T = XH'\mathbf{c}^T = \mathbf{0} \Leftrightarrow H'\mathbf{c}^T = \mathbf{0}$. En anden paritetstjeksmatrix er dermed på formen

$$H' = \begin{bmatrix} g(\alpha_1)^{-1} & \cdots & g(\alpha_n)^{-1} \\ \alpha_1 g(\alpha_1)^{-1} & \cdots & \alpha_n g(\alpha_n)^{-1} \\ \vdots & \cdots & \vdots \\ \alpha_1^{r-1} g(\alpha_1)^{-1} & \cdots & \alpha_n^{r-1} g(\alpha_n)^{-1} \end{bmatrix}. \tag{1.7}$$

For både H og H' gælder, at ønskes en paritetstjekmatrix med elementer i $\mathbb{F}_q$, så findes denne ved at erstatte hver enkelt indgang i matricen med vektoren af længde m i $\mathbb{F}_q$.

1.2.3 Eksempel:
Der arbejdes videre med eksempel 1.2.2. Det vides, at $\deg(g(x)) = 2$, dermed er en paritetstjekmatrix

$$
\begin{aligned}
H &= \begin{bmatrix} g(\alpha_1)^{-1} & \cdots & g(\alpha_n)^{-1} \\ \alpha_1 g(\alpha_1)^{-1} & \cdots & \alpha_n g(\alpha_n)^{-1} \\ \vdots & \cdots & \vdots \\ \alpha_1^{r-1} g(\alpha_1)^{-1} & \cdots & \alpha_n^{r-1} g(\alpha_n)^{-1} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{g(0)} & \frac{1}{g(1)} & \frac{1}{g(\alpha)} & \frac{1}{g(\alpha^2)} & \frac{1}{g(\alpha^3)} & \frac{1}{g(\alpha^4)} & \frac{1}{g(\alpha^5)} & \frac{1}{g(\alpha^6)} \\ \frac{0}{g(0)} & \frac{1}{g(1)} & \frac{\alpha}{g(\alpha)} & \frac{\alpha^2}{g(\alpha^2)} & \frac{\alpha^3}{g(\alpha^3)} & \frac{\alpha^4}{g(\alpha^4)} & \frac{\alpha^5}{g(\alpha^5)} & \frac{\alpha^6}{g(\alpha^6)} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 & \alpha^2 & \alpha^4 & \alpha^2 & \alpha & \alpha & \alpha^4 \\ 0 & 1 & \alpha^3 & \alpha^6 & \alpha^5 & \alpha^5 & \alpha^6 & \alpha^3 \end{bmatrix} \\
&= \begin{array}{c} \begin{matrix} 0 & 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \end{matrix} \\ \left[\begin{array}{cccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{array}\right] \end{array}
\end{aligned}
$$

Det bemærkes, at minimumsafstanden af koden er 5, da det er det mindste antal lineært afhængige søjler i paritetstjekmatricen. Dette kan blandt andet ses, da summen af de første fire søjler giver søjle seks. ◀

Når en paritetstjekmatrix kendes, kan en generatormatrix findes, da det vides, at $GH^T = \mathbf{0}$. Vektorerne i nulrummet for H modulo q udspænder rækkerummet for G. Dette illustreres nu i et eksempel.

1.2.4 Eksempel:
Eksempel 1.2.3 fortsættes. En generatormatrix beregnes i SAGE,

$$
G = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.
$$

Udregningerne kan ses i appendiks B.4. Kodeordene er altså

$$C = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \right\}.$$

Det ses, at koden er lineær, da den er lukket under addition, skalarmultiplikation med elementer fra $\mathbb{F}_2$ og indeholder nul-kodeordet. ◀

Nu er parametrene for Goppa-koder gennemgået, og nu gives en kort beskrivelse af, hvordan dekodingen af denne type koder foregår. Goppa-koder er som tidligere nævnt alternerende koder, og derfor kan dekodningsalgoritmerne for denne type koder benyttes, når Goppa-koder skal dekodes. I (MacWilliams m.fl., 1977) kapitel 12 paragraf 9 præsenteres en dekodning, der består af tre trin, hvor første trin er af finde syndromerne til koden. I andet trin findes *det fejlfindende polynomium* og *det fejlevaluerende polynomium* ved hjælp af Euklids algoritme. Det tredje og sidste trin i dekodningen består i at finde frem til placeringen og størrelsen af fejlene og derved rette dem.

Den kodetype, som McEliece oprindeligt foreslog til sit kryptosystem, er nu præsenteret. Som tidligere nævnt blev der i 1986 udviklet en ækvivalent udgave af McEliece kryptosystemet. Den primære forskel består i, at hvor McEliece kryptosystemet benytter en generatormatrix, så benytter det nye system, Niederreiter kryptosystemet, en paritetstjekmatrix.

1.3 Teori om Niederreiter kryptosystemet

Dette afsnit bygger på (Peters, 2011), (Berger m.fl., 2009) samt (Overbeck m.fl., 2009).

I 1986 udviklede Harald Niederreiter en ny udgave af McEliece kryptosystemet. Denne nye form for kryptering, der ses som den duale af McEliece kryptosystemet kaldes *Niederreiter kryptosystemet*. Dette følger, da en paritetstjekmatrix benyttes i stedet for en generatormatrix.

Niederreiter kryptosystemet har den samme sikkerhed som McEliece kryptosystemet, når der benyttes Goppa-koder, hvilket også betyder, at hvis en hacker kan bryde det ene kryptosystem, så vil den anden også kunne brydes. Forskellen i de to former for kryptosystemer er både strukturen af den offentlige nøgle, men også måden hvorpå der indkodes og dekodes.

En fordel ved Niederreiter kryptosystemet er blandet andet, at den offentlige nøgle er mindre end ved McEliece kryptosystemet, dog vil den offentlige nøgle have samme størrelse, hvis generatormatricen $\hat{G}$ i McEliece kryptosystemet er på systematisk form. Ved Niederreiter kryptosystemet har den offentlige nøgle størrelsen $(n-k)n$ bits, og når paritetstjekmatricen $\hat{H}$ er på systematisk form er størrelse reduceret til $(n-k)k$ bits. Dette skyldes, at det kun er nødvendigt at lagre den ikke-trivielle del af $\hat{H}$. Derimod

er ulempen at meddelelsen først skal repræsenteres ved en fejlvektor af længde n med vægt højest t, hvilket gør indkodningen og dekodningen langsommere ved brug af Niederreiter kryptosystemet (Overbeck m.fl., 2009). Ved afrundning af dette kapitel laves en sammenligning af, hvad henholdsvis indkodning og dekodning koster for McEliece og Niederreiter kryptosystemet ud fra nogle af de i projektet givne eksempler. Denne sammenligning viser også, at Niederreiter kryptosystemet er langsommere. Meddelelsen repræsenteres ved en fejlvektor, hvor meddelelsen $\mathbf{m}$ er de indgange i fejlvektoren $\mathbf{x}$, der er forskellige fra nul.

$$\begin{array}{ccc} \mathbb{F}_q^k & \longrightarrow & \mathbb{F}_q^n \\ \mathbf{m} & \longmapsto & \mathbf{x} \end{array}$$

Det er vigtigt, at meddelelsen i $\mathbb{F}_q^k$ har $k \leq t$, da det ellers ikke kan garanteres, at dekodningsalgoritmen for koden kan rette fejlene. Bemærk dog, at hvis der benyttes en listedekodningsalgoritme til dekodningen, så er det muligt at rette flere end t fejl under forudsætning af, at det accepteres, at output er en liste med mulige kodeord. Sandsynligheden for at en sådan liste vil indeholde mere end ét kodeord er dog minimal. Følgende algoritme benyttes til at generere den private og den offentlige nøgle.

Algoritme 4 Nøglegenerering i Niederreiter kryptosystemet

1: Vælg parametrene n, k og t således at $W_{q,n,d,t} \geq 2^\kappa$.
2: Bestem en tilfældig paritetstjekmatrix H for en $[n, k, d]$ kode C.
3: Bestem en tilfældig $n \times n$ permutationsmatrix P.
4: Bestem en tilfældig $(n-k) \times (n-k)$ invertibel matrix S.
5: Beregn $\hat{H} = SHP$.
6: **return** Offentlig nøgle $(\hat{H}, t)$ og privat nøgle $(S, H, P,$ dekodningsalgoritme for $C)$.

Den private nøgle består i Niederreiter kryptosystemet ligesom i McEliece kryptosystemet af en $n \times n$ permutationsmatrix. Ændringen i forhold til McEliece kryptosystemet er, at den private nøgle i Niederreiter kryptosystemet består af en tilfældig invertibel $(n-k) \times (n-k)$ matrix S, en paritetstjekmatrix H for en t-korrigerende kode med dimension k samt en kendt dekodningsalgoritme til denne kode.

Parametrene n, k og t er offentlige, og den offentlige nøgle i Niederreiter kryptosystemet består af $(n-k) \times n$ matricen $\hat{H} = SHP$. Det vises nu med et eksempel, hvordan den offentlige og private nøgle genereres.

1.3.1 Eksempel:

I dette eksempel arbejdes der med samme kode, som i eksempel 1.1.1. Dermed arbejdes der med en $RS_{13}[12, 3]$, og nu benyttes algoritme 4 til at danne den private og den offentlige nøgle. Det vides, at en Reed-Solomon-kode har en paritetstjekmatrix på formen

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{n-k} & x_2^{n-k} & \cdots & x_n^{n-k} \end{bmatrix},$$

og fra eksempel 1.1.1 haves $x_1 = 1$, $x_2 = 2$, $x_3 = 4$, $x_4 = 8$, $x_5 = 3$, $x_6 = 6$, $x_7 = 12$, $x_8 = 11$, $x_9 = 9$, $x_{10} = 5$, $x_{11} = 10$, og $x_{12} = 7$. Dermed bliver paritetstjekmatricen

$$H = \begin{bmatrix} 1 & 2 & 4 & 8 & 3 & 6 & 12 & 11 & 9 & 5 & 10 & 7 \\ 1 & 4 & 3 & 12 & 9 & 10 & 1 & 4 & 3 & 12 & 9 & 10 \\ 1 & 8 & 12 & 5 & 1 & 8 & 12 & 5 & 1 & 8 & 12 & 5 \\ 1 & 3 & 9 & 1 & 3 & 9 & 1 & 3 & 9 & 1 & 3 & 9 \\ 1 & 6 & 10 & 8 & 9 & 2 & 12 & 7 & 3 & 5 & 4 & 11 \\ 1 & 12 & 1 & 12 & 1 & 12 & 1 & 12 & 1 & 12 & 1 & 12 \\ 1 & 11 & 4 & 5 & 3 & 7 & 12 & 2 & 9 & 8 & 10 & 6 \\ 1 & 9 & 3 & 1 & 9 & 3 & 1 & 9 & 3 & 1 & 9 & 3 \\ 1 & 5 & 12 & 8 & 1 & 5 & 12 & 8 & 1 & 5 & 12 & 8 \end{bmatrix}.$$

Udregningerne er lavet i SAGE og kan ses i appendiks B.5. Bob skal nu i punkt 3 i algoritme 4 vælge en tilfældig 12×12 permutationsmatrix, i dette tilfælde vælges den samme som i eksempel 1.1.1. En tilfældig 9×9 invertibel matrix S vælges i punkt 4 i algoritme 4, og Bob vælger følgende.

$$S = \begin{bmatrix} 4 & 0 & 11 & 11 & 0 & 2 & 3 & 0 & 0 \\ 5 & 11 & 0 & 11 & 0 & 5 & 9 & 0 & 3 \\ 2 & 7 & 4 & 0 & 10 & 0 & 4 & 0 & 5 \\ 5 & 12 & 8 & 4 & 0 & 0 & 5 & 1 & 0 \\ 10 & 11 & 4 & 2 & 7 & 0 & 0 & 2 & 0 \\ 1 & 3 & 11 & 0 & 6 & 9 & 0 & 4 & 0 \\ 4 & 2 & 5 & 11 & 3 & 7 & 7 & 0 & 3 \\ 2 & 8 & 4 & 9 & 1 & 4 & 5 & 2 & 0 \\ 5 & 5 & 2 & 12 & 6 & 8 & 4 & 0 & 7 \end{bmatrix}$$

Nu beregner Bob, i punkt 5 i algoritme 4,

$$\hat{H} = SHP = \begin{bmatrix} 6 & 4 & 3 & 5 & 2 & 8 & 3 & 7 & 6 & 1 & 9 & 11 \\ 0 & 1 & 5 & 5 & 0 & 10 & 1 & 0 & 6 & 8 & 0 & 3 \\ 6 & 11 & 5 & 6 & 11 & 8 & 11 & 0 & 10 & 6 & 5 & 12 \\ 5 & 3 & 4 & 9 & 11 & 12 & 0 & 7 & 4 & 3 & 2 & 5 \\ 0 & 6 & 2 & 10 & 12 & 7 & 0 & 6 & 3 & 7 & 6 & 6 \\ 4 & 9 & 5 & 8 & 10 & 11 & 4 & 12 & 3 & 5 & 9 & 11 \\ 6 & 10 & 4 & 3 & 9 & 11 & 3 & 12 & 7 & 9 & 12 & 5 \\ 2 & 1 & 4 & 9 & 12 & 11 & 9 & 3 & 3 & 6 & 10 & 8 \\ 11 & 7 & 3 & 10 & 10 & 1 & 12 & 4 & 5 & 10 & 11 & 7 \end{bmatrix},$$

udregningerne kan ses i appendiks B.5. Bob offentliggør herefter $\hat{H}$ og $t = 4$, men beholder resten af informationerne selv. ◀

Når Alice ønsker at sende en meddelelse til Bob ved hjælp af Niederreiter kryptosystemet skal følgende algoritme 5 benyttes. En meddelelse i Niederreiter

kryptosystemet repræsenteres ved en fejlvektor i $\mathbb{F}_q^n$, med vægt w mindre end eller lig t. Niederreiter kryptosystemet kaldes netop den duale til McEliece kryptosystemet, da det er en fejlvektor i $\mathbb{F}_q^n$, der indkodes i stedet for en meddelelse i $\mathbb{F}_q^k$.

Algoritme 5 Indkodning i Niederreiter kryptosystemet

Input: En meddelelse $\mathbf{x} \in \mathbb{F}_q^n$, med vægt w og den offentlige nøgle, $\hat{H}$.
Output: En ciffertekst $\mathbf{s} \in \mathbb{F}_q^{n-k}$.

1: Beregn $\mathbf{s} = \hat{H}\mathbf{x}^T$.
2: **return** Cifferteksten $\mathbf{s}$.

Bemærk, at cifferteksten er et syndrom og dermed en søjlevektor. Der vises nu et eksempel på brugen af algoritme 5.

1.3.2 Eksempel:
Hvis Alice ønsker, at sende meddelelsen $\mathbf{m} = \begin{bmatrix} 4 & 7 & 9 \end{bmatrix}$, repræsenteres dette ved fejlvektoren $\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 4 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$. Hun benytter algoritme 5 og dermed den offentlige nøgle fra eksempel 1.3.1. Alice beregner nu

$$\mathbf{s} = \hat{H}\mathbf{x}^T = \begin{bmatrix} 10 \\ 2 \\ 1 \\ 3 \\ 3 \\ 3 \\ 0 \\ 2 \\ 5 \end{bmatrix},$$

og herefter sender hun denne ciffertekst til Bob. Udregningerne kan ses i appendiks B.6.◀

Bob modtager cifferteksten $\mathbf{s}$ fra Alice og for at dekode denne til meddelelsen $\mathbf{x}$ benyttes den følgende algoritme.

Algoritme 6 Dekodning i Niederreiter kryptosystemet

Input: En ciffertekst $\mathbf{s} = \hat{H}\mathbf{x}^T \in \mathbb{F}_q^{n-k}$ og den private nøgle (C, H, P, S).
Output: Meddelelsen $\mathbf{x}$.

1: Dan $\mathbf{z}$ som $H\mathbf{z}^T = S^{-1}\mathbf{s} = HP\mathbf{x}^T = H(\mathbf{x}P^T)^T$ ved brug af lineær algebra.
2: Benyt dekodningsalgoritmen for C til at finde kodeordet $\mathbf{z} - \mathbf{x}P^T$.
3: Brug lineær algebra til at finde $\mathbf{x}$, se ligning (1.8).
4: **return** Meddelelsen $\mathbf{x}$.

Algoritme 6 returnerer meddelelsen $\mathbf{x}$ når cifferteksten er indkodet med algoritme 5. Dette skyldes, at Bob kender matricerne H og S, hvormed det er muligt for ham at danne

vektoren $\mathbf{z} \in \mathbb{F}_q^n$ i linje 1 i algoritme 6, og det ses at $H(\mathbf{z} - \mathbf{x}P^T)^T = 0$. Da det gælder, at $H\mathbf{c}^T = 0$, hvis $\mathbf{c}$ er et kodeord i C, må der gælde, at $\mathbf{z} - \mathbf{x}P^T$ er et kodeord i koden C. Det vides, at $\mathbf{x}$ har vægt w, samt at $w \leq t$, og dermed kan dekodningsalgoritmen for koden benyttes til at finde $\mathbf{c} = \mathbf{z} - \mathbf{x}P^T$. Derfor kan $\mathbf{z}$ ses som ordet, der modtages med fejl, da $\mathbf{x}$ er fejlvektoren. Ud fra ovenstående haves, at

$$\mathbf{c} = \mathbf{z} - \mathbf{x}P^T \quad \Leftrightarrow \quad \mathbf{c}P = \mathbf{z}P - \mathbf{x} \quad \Leftrightarrow \quad \mathbf{x} = (\mathbf{z} - \mathbf{c})P. \tag{1.8}$$

Et eksempel på algoritme 6 vises nu.

1.3.3 Eksempel:
Bob har modtaget syndromet

$$\mathbf{s} = \begin{bmatrix} 10 & 2 & 1 & 3 & 3 & 3 & 0 & 2 & 5 \end{bmatrix}^T$$

fra Alice og han skal nu benytte algoritme 6 til at dekode. Til dette bruger han den private nøgle, han har genereret i eksempel 1.3.1.

Det første han skal gøre er at finde $\mathbf{z}$, der kan ses som det modtagede ord med fejl, hvilket er input i algoritme A.1. Dette gøres ved først at beregne $S^{-1}\mathbf{s}$.

$$S^{-1}\mathbf{s} = \begin{bmatrix} 2 & 6 & 4 & 8 & 12 & 2 & 3 & 11 & 3 \end{bmatrix}^T,$$

beregningerne af dette er gjort i SAGE og kan ses i appendiks B.7. Herefter skal Bob løse ligningssystemet

$$H\mathbf{z}^T = \begin{bmatrix} 2 & 6 & 4 & 8 & 12 & 2 & 3 & 11 & 3 \end{bmatrix}^T,$$

dette gøres i SAGE, beregningerne kan ses i appendiks B.7 og giver

$$\mathbf{z} = \begin{bmatrix} 4 & 9 & 11 & 3 & 4 & 9 & 5 & 6 & 5 & 0 & 0 & 0 \end{bmatrix}.$$

Da Bob ved, at koden er en Reed-Solomon-kode, kan han nu benytte en dekodningsalgoritme til at dekode $\mathbf{z}$ til et kodeord $\mathbf{c}$. Han benytter algoritmen fra appendiks A.1. Først beregner Bob $\ell_0 = n - t - 1 = 12 - 4 - 1 = 7$ og $\ell_1 = t = 4$, og derefter løser han det homogene lineære ligningssystem bestående af 12 ligninger med

13 ubekendte,

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 4 & 4 & 4 & 4 \\ 1 & 2 & 4 & 8 & 3 & 6 & 12 & 11 & 9 & 5 & 10 & 7 & 1 \\ 1 & 4 & 3 & 12 & 9 & 10 & 1 & 4 & 11 & 5 & 7 & 2 & 8 \\ 1 & 8 & 12 & 5 & 1 & 8 & 12 & 5 & 3 & 11 & 10 & 2 & 3 \\ 1 & 3 & 9 & 1 & 3 & 9 & 1 & 3 & 4 & 12 & 10 & 4 & 12 \\ 1 & 6 & 10 & 8 & 9 & 2 & 12 & 7 & 9 & 2 & 12 & 7 & 3 \\ 1 & 12 & 1 & 12 & 1 & 12 & 1 & 12 & 5 & 8 & 5 & 8 & 5 \\ 1 & 11 & 4 & 5 & 3 & 7 & 12 & 2 & 6 & 1 & 11 & 4 & 5 \\ 1 & 9 & 3 & 1 & 9 & 3 & 1 & 9 & 5 & 6 & 2 & 5 & 6 \\ 1 & 5 & 12 & 8 & 1 & 5 & 12 & 8 & 0 & 0 & 0 & 0 & 0 \\ 1 & 10 & 9 & 12 & 3 & 4 & 1 & 10 & 0 & 0 & 0 & 0 & 0 \\ 1 & 7 & 10 & 5 & 9 & 11 & 12 & 6 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ Q_{0,3} \\ Q_{0,4} \\ Q_{0,5} \\ Q_{0,6} \\ Q_{0,7} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ Q_{1,3} \\ Q_{1,4} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Han sætter matricen på reduceret-række-echelon-form

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 11 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

og vælger $Q_{1,4}$ som den frie variabel med værdien 1. Dermed opnår han

$$\begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ Q_{0,3} \\ Q_{0,4} \\ Q_{0,5} \\ Q_{0,6} \\ Q_{0,7} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ Q_{1,3} \\ Q_{1,4} \end{bmatrix} = \begin{bmatrix} 8 \\ 9 \\ 9 \\ 11 \\ 12 \\ 9 \\ 7 \\ 0 \\ 6 \\ 2 \\ 4 \\ 0 \\ 1 \end{bmatrix},$$

der giver polynomierne

$$Q_0(x) = 8 + 9x + 9x^2 + 11x^3 + 12x^4 + 9x^5 + 7x^6$$
$$Q_1(x) = 6 + 2x + 4x^2 + x^4.$$

Dermed opnås

$$f(x) = -\frac{Q_0(x)}{Q_1(x)} = 3 + 4x + 6x^2.$$

Kodeordet $\mathbf{c}$ beregnes nu ved at evluere $f(x)$ i alle x_i for $i = 1, \ldots, 12$.

$$\begin{aligned}\mathbf{c} &= \begin{bmatrix} f(1) & f(2) & f(4) & f(8) & f(3) & f(6) & f(12) & f(11) & f(9) & f(5) & f(10) & f(7) \end{bmatrix} \\ &= \begin{bmatrix} 0 & 9 & 11 & 3 & 4 & 9 & 5 & 6 & 5 & 4 & 6 & 0 \end{bmatrix}\end{aligned}$$

Det vides at $\mathbf{c} = \mathbf{z} - \mathbf{x}P^T$, hvormed $\mathbf{x} = (\mathbf{z} - \mathbf{c})P$, og Bob beregner nu,

$$\mathbf{x} = (\mathbf{z} - \mathbf{c})P = \begin{bmatrix} 0 & 0 & 0 & 4 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 9 \end{bmatrix},$$

hvilket netop er hvad Alice har sendt, hvormed Bob kan se, at meddelelsen er $\begin{bmatrix} 4 & 7 & 9 \end{bmatrix}$
Alle beregninger fra eksemplet kan ses i appendiks B.7. ◀

Som tidligere nævnt er ulempen ved Niederreiter kryptosystemet i forhold til McEliece kryptosystemet, at indkodning og dekodning er langsommere. Dette kan også ses ved sammenligning af eksempel 1.1.2 og 1.3.2 med hensyn til kryptering og eksempel 1.1.3 og 1.3.3 med hensyn til dekodning. Det er muligt at sammenligne, da det er den samme meddelelse og kode, der arbejdes med. Ses på indkodning ved McEliece kryptosystemet, hvor meddelelsen multipliceres med generatormatricen for koden, er der $2 \cdot 12 = 24$ additioner og $3 \cdot 12 = 36$ multiplikationer. Ved Niederreiter kryptosystemet er det paritetstjekmatricen og syndromer, som benyttes, og der er $11 \cdot 9 = 99$ additioner og $12 \cdot 9 = 108$ multiplikationer. Dermed ses det tydeligt, at Niederreiter kryptosystemet er langsommere. Når der ses på dekodning ses bort fra dekodningsalgoritmen for Reed-Solomon-koder, da det er den samme, der benyttes i begge eksempler, og input har samme længde. Ved dekodning gælder for både McEliece og Niedereiter kryptosystemerne, at der skal udføres et antal additioner og multiplikationer. Dog skal der ved Niederreiter kryptosystemet også løses et lineært ligningssystem, hvilket er dyrt. Dermed er Niederreiter dekodning langsommere end McEliece dekodning. Bemærk, at de eksempler, der er brugt til sammenligningen er forholdsvis små, og derfor vil forskellen i antallet af operationer generelt være betydeligt større.

McEliece og Niederreiters kryptosystemer er nu begge blevet præsenteret, og der har i kapitlet været særligt fokus på Goppa-koder, da det er disse, der er benyttet ved udviklingen af kryptosystemerne. Eksemplerne i kapitlet har taget udgangspunkt i Reed-Solomon-koder, da generaliseringen af disse i flere år var en god kandidat til kryptosystemerne, på grund af den hurtige ind- og dekodning. Når der arbejdes med Reed-Solomon-koder kan dekodningsalgoritmerne, der benyttes vælges til at være listedekodningsalgoritmer.

Dette gør, at at det er muligt at rette flere fejl, men ulempen er, at det risikeres at en liste med kodeord modtages. Der er dog en meget lille sandsynlighed for, at der opstår mere end et kodeord på listen. Som tidligere nævnt er Reed-Solomon-koder dog ikke sikre, og det er netop sikkerhed af systemerne, der er omdrejningspunktet i næste kapitel. Niederreiter kryptosystemet er som tidlige beskrevet det duale til McEliece kryptosystemet. De to kryptosystemer har derfor en del af de samme egenskaber, hvorfor det er valgt, at resten af projektet udelukkende bygger på McEliece kryptosystemet.

KAPITEL 2

Angreb på McEliece kryptosystemet

Dette kapitel bygger på (Berger m.fl., 2009), (Gaborit, 2004), (Sendrier, 2000), (Overbeck m.fl., 2009), (Misoczki, 2013), (Peters, 2009), (MacWilliams m.fl., 1977), (Roering, 2013) og (Márquez-Corbella m.fl., 2015). Der er overordnet set to typer af angreb på McEliece kryptosystemet, *meddelelsesangreb* og *nøgleangreb*. Meddelelsesangreb, som afsnit 2.1 omhandler, består i at dekode cifferteksten $\mathbf{y} = \mathbf{c} + \mathbf{e}$ uden at kende kodens type, altså blot ved hjælp af $\hat{G}$. Nøgleangreb omhandler at finde frem til kodens struktur ud fra $\hat{G}$, altså at finde G, hvilket behandles i afsnit 2.2.

Når der arbejdes med kryptering er begrebet *brute-force* centralt, hvilket vil sige at samtlige muligheder afprøves én for én. I henhold til at bryde McEliece kryptosystemet vil der være to forskellige fremgangsmåder, én for meddelelsesangreb og én for nøgleangreb. Ved meddelelsesangreb beregnes afstanden fra det modtagne ord til et hvert kodeord i koden, der er genereret af $\hat{G}$. Dette gøres indtil et kodeord med afstand mindre end eller lig t til det modtagede ord er fundet. Denne proces er meget langsom, og i McElieces oprindelige valg af kode er der $q^k = 2^{524} = 5.49 \cdot 10^{157}$ kodeord. En anden tilgang er nøgleangreb. Når der arbejdes med brute-force inden for dette er idéen at finde generatormatricen G for koden ud fra $\hat{G}$. Dette gøres ved at gætte på, hvordan matricerne G, S og P er valgt. Ved tilpas store parametre er de stort set umulige at finde. I følgende to afsnit gives bedre metoder til at bryde McEliece kryptosystem indenfor henholdsvis meddelelsesangreb og nøgleangreb.

2.1 Meddelelsesangreb

Meddelelsesangreb kaldes også dekodningsangreb, da idéen bag denne type angreb består i at dekode en ukendt kodetype. Dette svarer til et *generelt dekodningsproblem* eller ækvivalent et *generelt syndromdekodningsproblem*, der er NP-hard. Når der arbejdes med meddelelsesangreb kendes cifferteksten $\mathbf{y}$ samt den offentlige nøgle $(\hat{G}, t)$, og ud fra disse søges meddelelsen $\mathbf{m}$ eller fejlen $\mathbf{e}$ i cifferteksten. Ved generel dekodning arbejdes der med en generatormatrix, og meddelelsen findes direkte. Koden er genereret som følger

$$C = \langle G \rangle = \left\{ \mathbf{m}G \mid \mathbf{m} \in \mathbb{F}_q^k \right\},$$

hvor $G \in \mathbb{F}_q^{k\times n}$ er en generatormatrix, og med $C = \langle G \rangle$ menes, at koden er genereret af rækkerne i G. Dermed opnås følgende afbildning

$$\begin{aligned} \Phi: \quad \mathbb{F}_q^n \times \mathbb{F}_q^{k\times n} &\longrightarrow \mathbb{F}_q^k \\ (\mathbf{y}, G) &\longmapsto \mathbf{m}. \end{aligned}$$

I modsætning til generel dekodning arbejder generel syndromdekodning med en paritetstjekmatrix, og det er fejlvektoren, der findes direkte, mens meddelelsen herefter kan findes. Det vides, at koden er genereret som

$$C = \langle H \rangle^{\perp} = \left\{ \mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{c}H^T = \mathbf{0} \right\},$$

hvor $H \in \mathbb{F}_q^{(n-k)\times n}$ er en paritetstjekmatrix og med $C = \langle H \rangle^{\perp}$ menes at koden genereres af søjlerne i H. Generel syndromdekodning kan så beskrives ved følgende afbildning

$$\begin{aligned} \Psi: \quad \mathbb{F}_q^{n-k} \times \mathbb{F}_q^{(n-k)\times n} &\longrightarrow \mathbb{F}_q^n \\ (\mathbf{s}, H) &\longmapsto \mathbf{e}. \end{aligned}$$

Som tidligere nævnt er disse to former for dekodning ækvivalente, og begge problemer løses som regel ved brug af *information set dekoding*.

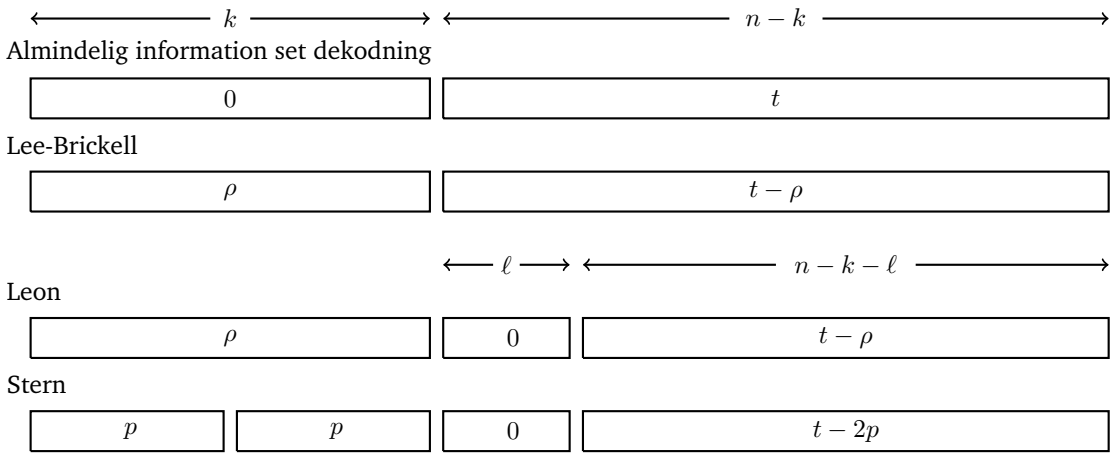
2.1.1 Information set dekodning

I 1962 foreslog Eugene Prange, som den første, brugen af information set dekodning (Prange, 1962). Dette er netop den type angreb, som McEliece foreslog i sin oprindelige artikel fra 1978. Han mente, at information set dekodning var det bedste angreb mod hans kryptosystem, og han arbejdede med en udgave, hvor det antages at de første k søjler er fejlfrie. Dette betegnes *Almindelig information set dekodning*. Nu defineres, hvad der menes med information set.

2.1.1 Definition:
Lad G være generatormatrix for en lineær $[n, k]$-kode og lad $\mathcal{I} = \{i_1, \ldots, i_k\}$ være en delmængde af $\{1, \ldots, n\}$ med kardinalitet k. Lad $G_{\mathcal{I}}$ være den $k \times k$ delmatrix af G, der er defineret ved søjleindeks svarende til elementerne i $\mathcal{I}$. Hvis $G_{\mathcal{I}}$ er invertibel, så er $\mathcal{I}$ et information set.

Når der arbejdes med information set dekodning, skal det bemærkes, at der findes flere forskellige udviklinger. Blandt andet har Pil Joong Lee, Ernest F. Brickell, Jeffrey S. Leon samt Jacques Stern videreudviklet almindelig information set dekodning. I 1988 udvidede Lee og Brickell denne type angreb, hvor de tillod, at der var ρ fejl i de k første indgange i kodeordet (Lee m.fl., 1988). Leon videreudviklede Lee-Brickell's algoritme ved at kræve, at der skal være ℓ positioner uden fejl udenfor information set (Leon, 1988). I 1989 udviklede Stern en algoritme, der fremstår som en blanding mellem Lee-Brickel og Leons algoritmer. Han benytter Lee-Brickell's idé om at tillade ρ fejl i information set, men han deler information set op i to dele, hver med $p = \frac{\rho}{2}$ fejl. Fra Leons metode benyttes konceptet

med stadig at kræve, at der er ℓ nul-indgange udenfor information set (Stern, 1989). De fire forskellige vægtfordelinger kan ses i figur 2.



Figur 2. Fordeling af fejl i almindelig information set dekodning, Lee-Brickell, Leon og Sterns algoritmer.

Stern arbejder som tidligere nævnt med en todeling af information set, og dette gør algoritmen hurtigere, da *birthday-angreb* benyttes til at finde kodeord med tilpas lav vægt. Birthday-angreb er udviklet som en løsning på birthday-paradokset, der omhandler, at finde sandsynligheden for, at to personer fra en gruppe på n personer har fødselsdag den samme dag. Det viser sig, at antages det, at der er 365 dage på et år, så skal der blot 23 personer til, for at sandsynligheden for at to personer har fødselsdag den samme dag er over 50%. Lad

$$A = \{ \text{ mindst to personer har samme fødselsdag } \}$$
$$B = \{ \text{ alle personer har forskellige fødselsdage } \}.$$

Sandsynligheden for at to personer har fødselsdag samme dag beregnes.

$$\begin{aligned} P(A) &= 1 - P(B) \\ &= 1 - \frac{365}{365} \cdot \frac{364}{365} \cdots \frac{344}{365} \cdot \frac{343}{365} \\ &= 1 - \frac{365 \cdot 364 \cdots 344 \cdot 343}{365^{23}} \\ &\approx 1 - 0.4927 \\ &= 0.5073 \end{aligned}$$

Sandsynligheden for at alle personer har forskellige fødselsdage beregnes ved hjælp af følgende tankegang. Den første person kan have fødselsdag alle 365 dage i året, den næste person har kun 364 mulige dage, da han ikke må have fødselsdag samme dag som første person. På tilsvarende måde har tredje person 363 mulige dage og så videre.

I 2002 fremlagde David Wagner en generalisering af dette problem. Generaliseringen består i, at der søges efter om k personer har fødselsdag samme dag, og han lavede en algoritme til at løse dette problem (Wagner, 2002). Han arbejdede dog udelukkende binært, men i 2009 generaliserede Lorenz Minder og Alistair Sinclair algoritmen til at gælde for $\mathbb{F}_q$ (Minder m.fl., 2012). I samtlige af algoritmerne der betragtes i figur 2 arbejdes over $\mathbb{F}_2$, men det er muligt at generalisere dem til at gælde for $\mathbb{F}_q$. I denne sammenhæng bliver birthday-problemet i Minder og Sinclairs version brugbar.

I dette projekt fokuseres på den generaliserede Stern algoritme, der er præsenteret af Christiane Peters i (Peters, 2011) samt i algoritme 7. Denne algoritme tager en generatormatrix G, en ciffertekst $\mathbf{y}$ samt et heltal t, der angiver den maksimale fejlvægt, som input. Da algoritmen dekoder ved hjælp af en generatormatrix G svarer det til et generelt dekodningsproblem, som tidligere beskrevet. Første trin i algoritmen er at vælge et information set $\mathcal{I}$ og restringere generatormatricens søjler til dem, der er indekseret med værdier fra $\mathcal{I}$. Denne noteres med $G_\mathcal{I}$ og er en $k \times k$ matrix. Da $G_\mathcal{I}$ er en delmatrix af G vides, at $G' = G_\mathcal{I}^{-1}G$ genererer samme kode som G. Bemærk, at søjler i G' med indeks fra $\mathcal{I}$ danner en $k \times k$ identitetsmatrix, da disse søjler svarer til $G_\mathcal{I}^{-1}G_\mathcal{I}$. Dermed kan G' sættes på systematisk form ved hjælp af søjlepermutation. Bemærk, at de indgange, der svarer til elementer i $\mathcal{I}$ er dem, der indeholder information. Den række i G', som indeholder et 1-tal på indgangen indekseret ved a_i i information set, noteres G_{a_i}.

Den generaliserede Stern algoritme, som ses i algoritme 7, søger at finde en fejlvektor $\mathbf{e}$ med vægt w, således at $\mathbf{y} - \mathbf{e}$ ligger i koden C. Lad $\mathbf{y}_\mathcal{I}$ betegne vektoren bestående af de indgange i cifferteksten $\mathbf{y}$ svarende til positionerne fra $\mathcal{I}$. Hvis $\mathbf{y}_\mathcal{I}$ er fejlfri, så noteres denne med $\mathbf{c}_\mathcal{I}$. Det gælder, at $\mathbf{c}_\mathcal{I} = \mathbf{m}G_\mathcal{I}$, da disse k positioner er de eneste, der indeholder information om meddelelsen. Her af følger

$$\mathbf{m} = \mathbf{c}_\mathcal{I} G_\mathcal{I}^{-1} \tag{2.1}$$

$$\mathbf{c} = (\mathbf{c}_\mathcal{I} G_\mathcal{I}^{-1})G \tag{2.2}$$

$$\mathbf{e} = \mathbf{y} - (\mathbf{c}_\mathcal{I} G_\mathcal{I}^{-1})G. \tag{2.3}$$

Algoritmen finder fejlvektoren $\mathbf{e}$ jævnfør ligning (2.3), og undersøger altså om $\mathbf{y}_\mathcal{I}$ indeholder fejl. Det antages, at der ligger $2p$ fejl i information set $\mathcal{I}$, og det kan ved permutation af søjlerne antages, at dette svarer til de k første søjler, se figur 2. Det er dog ikke en nødvendighed, at information set er de første k søjler, for at algoritmen terminerer. Idéen bag algoritmen er at udnytte fordelingen af fejlene, så derfor opdeles det nye $\mathbf{y}$, der noteres ved $\mathbf{y}_{temp} = \mathbf{y} - \mathbf{y}_\mathcal{I} G'$, i tre dele X, Y og Z med længde henholdsvis $\frac{k}{2}$, $\frac{k}{2}$ og $n-k$, når k er lige. I tilfældet hvor k er ulige vælges størrelsen på X til at være $\lfloor \frac{k}{2} \rfloor$, størrelsen på Y bliver $k - \lfloor \frac{k}{2} \rfloor$ mens størrelsen på Z forbliver den samme. Det undersøges nu, om X og Y begge har fejlvægt p og Z fejlvægt 0. Hvis dette er tilfældet, er der altså mindre end eller lig t fejl i $\mathbf{y}_t emp$, og det vides dermed at den korrekte fejlvektor er fundet, og ud fra denne kan meddelelsen $\mathbf{m}$ findes.

Algoritme 7 Generaliseret Stern

Input: Den offentlige nøgle fra McEliece kryptosystemet, en generatormatrix $\hat{G} \in \mathbb{F}_q^{k \times n}$ for en t-korrigerende kode $C \in \mathbb{F}_q^n$ og et heltal $t \geq 0$, samt en ciffertekst $\mathbf{y} \in \mathbb{F}_q^n$.

Output: En fejlvektor $\mathbf{e} \in \mathbb{F}_q^n$ med vægt $w \leq t$, hvor $\mathbf{y} - \mathbf{e} \in C$.

1: **Repeat**
2: Vælg et information set $\mathcal{I}$.
3: Beregn $G' = \hat{G}_{\mathcal{I}}^{-1}\hat{G}$.
4: Beregn $\mathbf{y}_{temp} = \mathbf{y} - \mathbf{y}_{\mathcal{I}}G'$.
5: Vælg en uniform tilfældig delmængde $X \subseteq \mathcal{I}$ med størrelse $\left\lfloor \frac{k}{2} \right\rfloor$.
6: Definer $Y = I \backslash X$.
7: Vælg en ℓ størrelse delmængde Z af $\{1, \ldots, n\}$ hvor $0 \leq \ell \leq n - k$.
8: **For each** uniform tilfældig størrelse $0 \leq p \leq t$ delmængde $A = \{a_1, a_2, ldots, a_p\} \subseteq X$ og **for each** $\widetilde{\mathbf{m}} = (\widetilde{m}_1, \ldots, \widetilde{m}_p) \in (\mathbb{F}_q^*)^p$, hvor $\widetilde{m}_i$ betegner mængden af elementer i $(\mathbb{F}_q^*)$. Betragt de p rækker $\mathbf{r}_i = \widetilde{m}_i G_{a_i}$, hvor G_{a_i} er som beskrevet på forrige side, og $\widetilde{m}_i$ antager $q - 1$ forskellige værdier. Beregn

$$\phi_{\widetilde{\mathbf{m}}}(A) = \mathbf{y}_{temp} - \sum_i \mathbf{r}_i,$$

restringeret til ℓ søjler, indekseret ved Z.
9: **For each** uniform tilfældig størrelse $0 \leq p \leq t$ delmængde $B = \{b_1, b_2, \ldots, b_p\} \subseteq Y$ og **for each** $\mathbf{m}' = (m'_1, \ldots, m'_p) \in (\mathbb{F}_q^*)^p$, hvor m'_j betegner mængden af elementer i $(\mathbb{F}_q^*)$. Betragt de p rækker $\mathbf{r}'_j = m'_j G_{b_j}$, hvor og m'_j antager $q - 1$ forskellige værdier og dan

$$\psi_{\mathbf{m}'}(B) = \sum_j \mathbf{r}_j,$$

restringeret til ℓ søjler, indekseret ved Z.
10: **For each** par (A, B), samt $\widetilde{m}$ og m', hvor det gælder, at

$$\phi_{\widetilde{m}}(A) = \psi_{m'}(B)$$

11: Beregn $\mathbf{e} = \mathbf{y}_{temp} - \sum_i \widetilde{m}_i G_{a_i} + \sum_j m'_j G_{b_j}$.
12: **Until** Fejlvektoren $\mathbf{e}$ har vægt mindre end eller lig t.
13: **return** Fejlvektoren $\mathbf{e}$.

Algoritme 7 har dog en meget stor kompleksitet. Det er i (Peters, 2011) vist, at et enkelt gennemløb af algoritmen koster følgende antal operationer.

$$\begin{aligned} \text{cost} = &(n-k)^2 (n+k) + ((\kappa - p + 1) + 2N(q-1)^p)\,\ell \\ &+ \frac{q}{q-1}(t - 2p + 1)2p \left(1 + \frac{q-2}{q-1}\right) \frac{N^2 (q-1)^{2p}}{q^\ell}. \end{aligned} \tag{2.4}$$

Hvis det antages, at k er lige vil $\mathcal{I}$ blive delt i to disjukte mængder af størrelse $\frac{k}{2}$, og i dette tilfælde vil $\kappa = \frac{k}{2}$ og $N = \binom{k/2}{p}$. Denne cost er en anden måde at notere workfactor. Antallet af gennemløb, det kan risikeres, at algoritmen skal igennem afhænger af, hvor

mange forskellige information set, der kan dannes. Der skal være k indgange i information set, og når første indgang er valgt, kan den næste vælges ud fra $n-1$ indgange, dette fortsættes indtil, der er k indgange i information set. På denne måde bliver det maksimale antal iterationer, der kan forekomme af algoritmen

$$\frac{n!}{(n-k)!},$$

og dermed kan den samlede kompleksitet maksimalt blive

$$\begin{aligned}\text{cost} &= \left((n-k)^2(n+k) + ((\kappa - p + 1) + 2N(q-1)^p)\,\ell\right. \\ &\quad + \frac{q}{q-1}(t-2p+1)2p\left(1+\frac{q-2}{q-1}\right)\frac{N^2(q-1)^{2p}}{q^\ell}) \cdot \frac{n!}{(n-k)!} \\ &\geq \left((n-k)^2(n+k)\right)\frac{n!}{(n-k)!}\end{aligned}$$

Dette er meget stort, hvilket også er forventeligt, da det skal være svært at bryde McEliece kryptosystemet. Nu gives et eksempel, der illustrerer brugen af algoritme 7, samt at kompleksitetet for selv små parametre er voldsom.

2.1.2 Eksempel:

I dette eksempel forsøges at bryde krypteringen og finde frem til meddelelsen $\mathbf{m}$ ud fra den offentlige nøgle, der i McEliece kryptosystemet består af en generatormatrix, i dette eksempel givet ved

$$\hat{G} = \begin{bmatrix} 0 & 5 & 6 & 4 & 6 & 5 \\ 2 & 3 & 2 & 5 & 1 & 5 \end{bmatrix},$$

samt $t = 2$. Derudover haves cifferteksten $\mathbf{y} = \begin{bmatrix} 0 & 3 & 3 & 2 & 5 & 6 \end{bmatrix}$ samt, at der arbejdes over legemet $\mathbb{F}_7$. Alt dette benyttes som input i algoritme 7. I punkt 2 vælges et information set, hvilket skal have længde $k = 2$, og det vælges her til $\mathcal{I} = \{1, 2\}$. Herefter skal G' dannes i punkt 3. Beregningerne af dette er lavet i SAGE, og kan ses i appendiks B.8. Det opnås, at

$$\begin{aligned} G' &= \hat{G}_{\mathcal{I}}^{-1}\hat{G} \\ &= \begin{bmatrix} 1 & 0 & 2 & 2 & 5 & 1 \\ 0 & 1 & 4 & 5 & 4 & 1 \end{bmatrix}. \end{aligned}$$

Herefter skal der i punkt 4 i algoritmen dannes $\mathbf{y}_{temp}$, og beregningerne af dette kan ses i appendiks B.8, og det opnås

$$\begin{aligned} \mathbf{y}_{temp} &= \mathbf{y} - \mathbf{y}_{\mathcal{I}}G' \\ &= \begin{bmatrix} 0 & 0 & 5 & 1 & 0 & 3 \end{bmatrix}. \end{aligned}$$

I punkt 5 og 6 vælges henholdsvis en delmængde $X \subseteq \mathcal{I}$, og en delmængde $Y = \mathcal{I} \backslash X$. Delmængderne vælges sådan, at $X = \{1\}$, hvormed $Y = \{2\}$. Nu skal ℓ og en delmængde Z bestemmes i punkt 7. Da $0 \leq \ell \leq 4$, kan ℓ vælges til 2, og $Z = \{4, 5\}$. Herefter skal p vælges, så $0 \leq p \leq 2$, og i dette gennemløb af algoritmen vælges $p = 1$, hvormed en

delmængde af X vælges som $A = \{a_1\} = \{1\}$. De p rækker $\mathbf{r}_i$ skal nu betragtes for alle mulige $\widetilde{m}$. I dette eksempel vil det sige $\widetilde{m}_1 = 1, \ldots, 6$, og herefter skal $\phi_{\widetilde{m}}(A)$ beregnes, dette ses i tabel 2.

$\widetilde{m}_1$	$\mathbf{r}_1$	$\phi_{\widetilde{m}}(A) = \mathbf{y} - \mathbf{r_1}$
1	$\begin{bmatrix}1\end{bmatrix}\begin{bmatrix}1&0&2&2&5&1\end{bmatrix} = \begin{bmatrix}1&0&2&2&5&1\end{bmatrix}$	$\begin{bmatrix}1&0\end{bmatrix} - \begin{bmatrix}2&5\end{bmatrix} = \begin{bmatrix}6&2\end{bmatrix}$
2	$\begin{bmatrix}2\end{bmatrix}\begin{bmatrix}1&0&2&2&5&1\end{bmatrix} = \begin{bmatrix}2&0&4&4&3&2\end{bmatrix}$	$\begin{bmatrix}1&0\end{bmatrix} - \begin{bmatrix}4&3\end{bmatrix} = \begin{bmatrix}4&4\end{bmatrix}$
3	$\begin{bmatrix}3\end{bmatrix}\begin{bmatrix}1&0&2&2&5&1\end{bmatrix} = \begin{bmatrix}3&0&6&6&1&3\end{bmatrix}$	$\begin{bmatrix}1&0\end{bmatrix} - \begin{bmatrix}6&1\end{bmatrix} = \begin{bmatrix}2&6\end{bmatrix}$
4	$\begin{bmatrix}4\end{bmatrix}\begin{bmatrix}1&0&2&2&5&1\end{bmatrix} = \begin{bmatrix}4&0&1&1&6&4\end{bmatrix}$	$\begin{bmatrix}1&0\end{bmatrix} - \begin{bmatrix}1&6\end{bmatrix} = \begin{bmatrix}0&1\end{bmatrix}$
5	$\begin{bmatrix}5\end{bmatrix}\begin{bmatrix}1&0&2&2&5&1\end{bmatrix} = \begin{bmatrix}5&0&3&3&4&5\end{bmatrix}$	$\begin{bmatrix}1&0\end{bmatrix} - \begin{bmatrix}3&4\end{bmatrix} = \begin{bmatrix}5&3\end{bmatrix}$
6	$\begin{bmatrix}6\end{bmatrix}\begin{bmatrix}1&0&2&2&5&1\end{bmatrix} = \begin{bmatrix}6&0&5&5&2&6\end{bmatrix}$	$\begin{bmatrix}1&0\end{bmatrix} - \begin{bmatrix}5&2\end{bmatrix} = \begin{bmatrix}3&5\end{bmatrix}$

Tabel 2. Udregningerne fra punkt 8 i algoritme 7

I punkt 9 i den generaliserede stern algoritme skal lignende beregninger, som i punkt 8 laves. Her skal en delmængde B af Y med størrelse p vælges. De p rækker $\mathbf{r}'_j$ skal nu betragtes for alle mulige m', og herefter skal $\psi_{m'}(B)$ beregnes, dette ses i tabel 3.

m'_1	$\mathbf{r}_1$	$\psi_{m'}(B) = \mathbf{r_1}'$
1	$\begin{bmatrix}1\end{bmatrix}\begin{bmatrix}0&1&4&5&4&1\end{bmatrix} = \begin{bmatrix}0&1&4&5&4&1\end{bmatrix}$	$\begin{bmatrix}5&4\end{bmatrix}$
2	$\begin{bmatrix}2\end{bmatrix}\begin{bmatrix}0&1&4&5&4&1\end{bmatrix} = \begin{bmatrix}0&2&1&3&1&2\end{bmatrix}$	$\begin{bmatrix}3&1\end{bmatrix}$
3	$\begin{bmatrix}3\end{bmatrix}\begin{bmatrix}0&1&4&5&4&1\end{bmatrix} = \begin{bmatrix}0&3&5&1&5&3\end{bmatrix}$	$\begin{bmatrix}1&5\end{bmatrix}$
4	$\begin{bmatrix}4\end{bmatrix}\begin{bmatrix}0&1&4&5&4&1\end{bmatrix} = \begin{bmatrix}0&4&2&6&2&4\end{bmatrix}$	$\begin{bmatrix}6&2\end{bmatrix}$
5	$\begin{bmatrix}5\end{bmatrix}\begin{bmatrix}0&1&4&5&4&1\end{bmatrix} = \begin{bmatrix}0&5&6&4&6&5\end{bmatrix}$	$\begin{bmatrix}4&6\end{bmatrix}$
6	$\begin{bmatrix}6\end{bmatrix}\begin{bmatrix}0&1&4&5&4&1\end{bmatrix} = \begin{bmatrix}0&6&3&2&3&6\end{bmatrix}$	$\begin{bmatrix}2&3\end{bmatrix}$

Tabel 3. Udregningerne fra punkt 9 i algoritme 7

Nu skal det i punkt 10 tjekkes, om der er par, hvor det gælder, at $\phi_{\widetilde{m}}(A) = \psi_{m'}(B)$. Det ses, at der er ét par, hvor dette gælder, nemlig $\phi_{\widetilde{m}}(A)$ med $\widetilde{m}_i = 1$ samt $\psi_{m'}(B)$ med $m'_j = 4$. Dette par er med markeret med rød i tabel 2 og tabel 3. Nu skal fejlvektoren for dette par dannes i punkt 11.

$$\begin{aligned}
\mathbf{e} &= y - \sum_i \widetilde{m}_i G_{a_i} + \sum_j m'_j G_{b_j} \\
&= \begin{bmatrix}0&0&5&1&0&3\end{bmatrix} - \left(\begin{bmatrix}1&0&2&2&5&1\end{bmatrix} + \begin{bmatrix}0&4&2&6&2&4\end{bmatrix}\right) \\
&= \begin{bmatrix}6&3&1&0&0&5\end{bmatrix}
\end{aligned}$$

Denne fejlvektor har vægt 4, hvilket er større end $t = 2$, hvormed det ikke er det rigtige information set, der er valgt. Derfor gennemløbes repeatløkken igen, og der skal vælges et

nyt information set. Dette gøres indtil outputtet bliver en fejlvektor med vægt mindre end eller lig t. I dette eksempel er der $6 * 5 = 30$ mulige information set, og der er derfor risiko for at repeatløkken skal gennemløbes 30 gange for at opnå den rigtige fejlvektor.

Der vælges nu et nyt information set $\mathcal{I} = \{2, 5\}$, og G' og det nye y dannes, se appendiks B.8 for disse beregninger. Det opnås, at

$$G' = \begin{bmatrix} 2 & 1 & 1 & 2 & 0 & 3 \\ 3 & 0 & 6 & 6 & 1 & 3 \end{bmatrix}$$
$$\mathbf{y} = \begin{bmatrix} 0 & 0 & 5 & 1 & 0 & 3 \end{bmatrix}.$$

Delmængderne X og Y vælges til $X = \{2\}$ og $Y = \{5\}$. Nu skal ℓ og en delmængde Z bestemmes, og ℓ vælges som før til 2 og $Z = \{1, 3\}$. Som i tidligere gennemløb af algoritmen vælges $p = 1$, og A vælges til $A = \{2\}$. De p rækker $\mathbf{r}_i$ betragtes for alle mulige $\widetilde{m}$, og $\phi_{\widetilde{m}}(A)$ beregnes herefter. Disse beregninger kan ses i appendiks B.8 i tabel 8. Beregningerne for punkt 9 i algoritmen, hvor $\psi_{m'}(B)$ beregnes kan også ses i appendiks B.8, hvor $B = \{5\}$. En samling af disse beregninger kan ses i tabel 4.

$\widetilde{m}_1 = m'_1$	$\phi_{\widetilde{m}}(A)$	$\psi_{m'}(B)$
1	$\begin{bmatrix} 5 & 4 \end{bmatrix}$	$\begin{bmatrix} 3 & 6 \end{bmatrix}$
2	$\begin{bmatrix} 3 & 3 \end{bmatrix}$	$\begin{bmatrix} 6 & 5 \end{bmatrix}$
3	$\begin{bmatrix} 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 2 & 4 \end{bmatrix}$
4	$\begin{bmatrix} 6 & 1 \end{bmatrix}$	$\begin{bmatrix} 5 & 3 \end{bmatrix}$
5	$\begin{bmatrix} 4 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \end{bmatrix}$
6	$\begin{bmatrix} 2 & 6 \end{bmatrix}$	$\begin{bmatrix} 4 & 1 \end{bmatrix}$

Tabel 4. Sammenligning af $\phi_{\widetilde{m}}(A)$ og $\psi_{m'}(B)$

Det ses, at der er ét par, hvor $\phi_{\widetilde{m}}(A) = \psi_{m'}(B)$. Dette er ved $\widetilde{m}_i = 3$ og $m'_j = 5$. Fejlvektoren $\mathbf{e}$ bliver dermed

$$\begin{aligned} \mathbf{e} &= \begin{bmatrix} 0 & 0 & 5 & 1 & 0 & 3 \end{bmatrix} - \left(\begin{bmatrix} 6 & 3 & 3 & 6 & 0 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 2 & 2 & 5 & 1 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0 & 4 & 0 & 0 & 2 & 0 \end{bmatrix}. \end{aligned}$$

Denne fejlvektor har vægt 2, hvilet nøjagtigt er t. Dette betyder altså, at det er den rigtige fejlvektor, der er fundet. Kodeordet uden fejl kan nu beregnes

$$\begin{aligned} \mathbf{c} &= \mathbf{y} - \mathbf{e} \\ &= \begin{bmatrix} 0 & 3 & 3 & 2 & 5 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 4 & 0 & 0 & 2 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 6 & 3 & 2 & 3 & 6 \end{bmatrix}. \end{aligned}$$

Nu undersøges, om kodeordet $\mathbf{c}$ ligger i koden C. For at undersøge dette benyttes, at kodeordet $\mathbf{c}$ er i koden C hvis og kun hvis $H\mathbf{c}^T = \mathbf{0}$. Udregningerne ses i appendiks B.8.

$$H\mathbf{c}^T = \begin{bmatrix} 5 & 3 & 1 & 0 & 0 & 0 \\ 5 & 2 & 0 & 1 & 0 & 0 \\ 2 & 3 & 0 & 0 & 1 & 0 \\ 6 & 6 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 6 \\ 3 \\ 2 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Kodeordet svarer til $\mathbf{c} = (\mathbf{c}_{\mathcal{I}} G_{\mathcal{I}}^{-1})G$, hvor $\mathbf{y}_{\mathcal{I}} = \mathbf{c}_{\mathcal{I}}$, da $\mathbf{y}_{\mathcal{I}}$ er fejlfri jævnfør ligning (2.2). Vektoren $\mathbf{c}_{\mathcal{I}}$ bliver altså

$$\mathbf{c}_{\mathcal{I}} = \begin{bmatrix} 3 & 5 \end{bmatrix} - \begin{bmatrix} 4 & 2 \end{bmatrix} = \begin{bmatrix} 6 & 3 \end{bmatrix}.$$

Meddelelsen kan beregnes jævnfør ligning (2.1)

$$\mathbf{m} = \mathbf{c}_{\mathcal{I}} G_{\mathcal{I}}^{-1} = \begin{bmatrix} 6 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 4 & 0 \end{bmatrix}.$$

Da det vides, at $\mathbf{c} \in C$ med afstand t til cifferteksten $\mathbf{y}$, er det den korrekte meddelelse, der er fundet. Der ses nu på, hvad ét gennemløb af algoritmen koster, når parametrene er som i dette eksempel, og til dette benyttes ligning (2.4). Udregningen bliver dermed

$$\begin{aligned} \text{cost} &= (6-2)^2 (6+2) + \left(\left(\frac{2}{2} - 1 + 1 \right) + 2 \cdot \binom{2/2}{1} \cdot (7-1)^1 \right) \cdot 2 \\ &\quad + \frac{7}{7-1}(2 - 2 \cdot 1 + 1)2 \cdot 1 \left(1 + \frac{7-2}{7-1} \right) \frac{\binom{2/2}{1}^2 (7-1)^{2 \cdot 1}}{7^2} \\ &= 4^2 \cdot 8 + (1 + 2 \cdot 6) \cdot 2 + \frac{7}{6} \cdot 2 \left(1 + \frac{5}{6} \right) \frac{6^2}{7^2} \\ &\approx 157. \end{aligned}$$

Det kan som tidligere nævnt risikeres at algoritmen skal gennemløbes 30 gange, hvormed den samlede omkostning kan risikere at blive $157 \cdot 30 = 4710$ operationer. Det kan derfor også ses, at algoritmen er meget omkostningsfuld selv for meget små parametre. ◀

Meddelelsesangreb er én type af angreb, der er benyttet til at bryde McEliece kryposystemet. Der findes som tidligere nævnt en anden type angreb, kaldet nøgleangreb. Denne type forsøger i modsætning til meddelelsesangreb ikke at finde meddelelsen direkte, men derimod at finde kodens struktur for dermed at kunne dekode og finde meddelelsen.

2.2 Nøgleangreb

Nøgleangreb kaldes også strukturelle angreb. Ved nøgleangreb forsøger en udefrakommende at opnå privat information ud fra den offentlige nøgle. Dette gøres ved at finde en struktur i den ellers ukendte kode, hvilket svarer til at finde G ud fra $\hat{G}$. Nøgleangreb har blandt andet vist sig at kunne bryde McEliece kryptosystemet for generaliserede

Reed-Solomon-koder. I forsøget på at bryde McEliece kryptosystemet ved et nøgleangreb benyttes *The support splitting algorithm*. Konceptet bag algoritmen er at undersøge, om to koder C og C' er ækvivalente under isometri. I projektet arbejdes med semi-lineære isometrier, der afbilder et underrum af et vektorrum ind i et andet. I den forbindelse arbejdes med *support*.

2.2.1 Definition:
Lad C være en kode, hvorom deet gælder at ingen indgange er nul for alle kodeord. Lad desuden C være en kode med længde n, så er kodens support $\mathcal{J} = \{1, \ldots, n\}$ en ordnet mængde med kardinalitet n, der benyttes til indeksering af indgangene i kodeordene. Et kodeord $\mathbf{c}_j$'s support er den delmængde $\mathcal{J}_{\mathbf{c}_j}$ af $\mathcal{J}$, der består af ikkenul-indgangene i $\mathbf{c}_j$.

Nu gives et eksempel på support.

2.2.2 Eksempel:
Goppa-koden fra eksempel 1.2.4 har følgende kodeord.

$$C = \left\{ \begin{array}{l} \mathbf{c}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{c}_2 = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ \mathbf{c}_3 = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \\ \mathbf{c}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \right\}.$$

Kodeordenes support er givet ved

j	$\mathcal{J}_{\mathbf{c}_j}$
1	$\emptyset$
2	$\{3, 4, 5, 6, 7, 8\}$
3	$\{1, 2, 5, 7, 8\}$
4	$\{1, 2, 3, 4, 6\}$

Tabel 5. Support til kodeordene i C.

Det ses, desuden fra tabel 5, at kodens support må være $\mathcal{J} = \{1, 2, 3, 4, 5, 6, 7, 8\}$. ◄

Når support er defineret, kan dette benyttes til at beskrive de semi-lineære isometrier i Hamming-rummet $\mathbb{F}_q^n$, hvorfor afstanden mellem kodeord bevares. Support benyttes til de semi-lineære isometrier på følgende måde

$$\begin{array}{rccc} \varphi: & \mathbb{F}_q^n & \longrightarrow & \mathbb{F}_q^n \\ & (x_i)_{i \in \mathcal{J}} & \longmapsto & \left(\lambda_i \pi \left(x_{\sigma^{-1}(i)}\right)\right)_{i \in \mathcal{J}}, \end{array}$$

hvor x_i er i'te position i et kodeord $\mathbf{x}$ og $\mathcal{J}$ er kodens support. Afbildningen φ består af en permutation σ af de positioner af kodeordene, der er en del af supporten $\mathcal{J}$.

Yderligere indgår en skalarmultiplikation med $\lambda = (\lambda_1, \ldots, \lambda_n) \in \left(\mathbb{F}_q^*\right)^n$ samt en automorfi $\pi : \mathbb{F}_q \longrightarrow \mathbb{F}_q$. Da der udelukkende arbejdes med lineære koder i dette projekt, er der ingen tab af generalitet ved kun at betragte semi-lineære isometrier fremfor isometrier generelt. Dette skyldes, at når der eksisterer en isometri, der afbilder C over i C', så eksisterer også en semi-lineær isometri, da lineære koder er defineret over vektorrum. Nu ses på ækvivalensen af sådanne to koder.

2.2.3 Definition:
Hvis koden C afbildes over i koden C', eller omvendt, ved en semi-lineær isometri, siges C og C' at være ækvivalente. Hvis der eksisterer en permutation σ således at

$$C' = \sigma(C) = \left\{ \left(x_{\sigma^{-1}(i)}\right)_{i \in \mathcal{J}} \mid (x_i)_{i \in \mathcal{J}} \in C \right\}$$

siges C og C' at være permutationsækvivalente.

Bemærk, at i det binære tilfælde svarer en semi-lineær isometri til en permutation, da λ_i altid har værdien 1 og den eneste automorfi π, der afbilder $\mathbb{F}_2$ ind i sig selv er identitetsafbildningen. Dermed gælder i dette tilfælde, at når to koder er permutationsækvivalente, så er de også ækvivalente og omvendt.

Ækvivalensen mellem to koder giver, at to ækvivalente koder kan rette samme antal fejl. Konkret betyder dette, at hvis en kode C er t-korrigerende, og der haves en dekodningsalgoritme til denne. Så vil en t-korrigerende dekodningsalgoritme for C' svare til først at afbilde C' ind i C ved φ, og dernæst benytte dekodningsalgoritmen for C for tilsidst at afbilde C tilbage ind i C' ved φ^{-1}.

Når the support splitting algorithm benyttes er konceptet som tidligere nævnt at finde ud af om to koder er ækvivalente under isometri. Dette gøres ved at se på generatormaticerne G og G' for henholdsvis C og C' og undersøge om disse generatormatricer er permutationsækvivalente. Erez Petrank og Ron M. Roth har vist, at dette problem ikke er NP-komplet, men dog at det er lige så svært at løse som *graf isomorfi problemet*. Dette betyder, at det ikke vides, om det er muligt at løse problemet i polynomiel tid (Petrank m.fl., 1997).

I the support splitting algorithm benyttes *signatur*, men før dette kan defineres, er det nødvendigt at definere, hvad det vil sige at en afbildning er *invariant*.

2.2.4 Definition:
En afbildning ν siges at være invariant, hvis to permutaionsækvivalente koder C og C' angiver samme værdier, og dermed gælder

$$\nu(C) = \nu(C').$$

Afbildninger, der er invariante over heltal, kan for eksempel være længden, antallet af kodeord samt minimumsafstanden. Lad $\alpha_i = |\{\mathbf{c} \in C \mid w(\mathbf{c}) = i\}|$ være *vægtfordelingen* for koden C. Så er *vægtenumeratoren*

$$\mathcal{W}_C(x) = \sum_{i=0}^{n} \alpha_i x^i$$

også invariant, hvilket betyder, at $\mathcal{W}_C(x) = \mathcal{W}_{C'}(x)$. Det kan udnyttes, at disse afbildninger er invariante, og dermed kan det bestemmes om to koder er permutationsækvivalente. Det ses, at to koder med forskellig vægtenumerator ikke kan være permutationsækvivalente. Hvis der derimod ses på to koder med samme vægtenumerator, er dette dog ikke tilstrækkeligt for at sige, at de to koder er permutationsækvivalente. Dette skyldes, at der eksisterer koder, der har samme vægtfordeling, men som ikke er permutationsækvivalente. Nu gives et eksempel på dette.

2.2.5 Eksempel:

I dette eksempel arbejdes med to $[6, 3]$-koder over $\mathbb{F}_2$, C og C'. Generatormatricerne for C og C' er henholdsvis

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad \text{og} \qquad G' = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

De to koder er altså

$$C = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \right.$$
$$\left. \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \right.$$
$$\left. \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \right\}$$

og

$$C' = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \right.$$
$$\left. \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \right.$$
$$\left. \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \right\}.$$

Nu betragtes kodernes vægtfordeling α_i, da det ønskes at undersøge, om koderne har samme vægtenumerator. Vægtfordelingen af de to koder er opstillet i tabel 6.

	C	C'
α_0	1	1
α_1	0	0
α_2	3	3
α_3	0	0
α_4	3	3
α_5	0	0
α_6	1	1

Tabel 6. Vægtfordeling for koderne C og C'.

Det ses altså, at koderne har samme vægtenumerator

$$\begin{aligned}\mathcal{W}_C(x) = \mathcal{W}_{C'}(x) &= \sum_{i=0}^{6} \alpha_i x^i \\ &= 1x^0 + 0x^1 + 3x^2 + 0x^3 + 3x^4 + 0x^5 + 1x^6 \\ &= 1 + 3x^2 + 3x^4 + x^6.\end{aligned}$$

Nu undersøges om de to koder er permutationsækvivalente, hvilket gøres ved at betragte kodeordene med vægt 2 i både C og C'. Det er ikke muligt at permutere kodeordenes positioner i C, sådan at kodeordnene i C' opnås. Dette ses, da der for de tre kodeord med vægt 2 i C gælder, at der for hvert indeks kun er ét af kodeordene, der har en ikkenulindgang Dette gør sig ikke gældende for de tre kodeord med vægt 2 i C', hvor der ved indeks 2 og 5 er to kodeord med indgange forskellige fra 0. Dermed er de to koder ikke permutationsækvivalente på trods af, at de har den samme vægtenumerator. ◀

Det er dog meget sjældent at to koder med samme vægtenumerator ikke er permutationsækvivalente. For at garantere at to koder er permutationsækvivalente ved hjælp af invarians, herunder vægtenumeratoren, er det nødvendigt at benytte begrebet *udprikket kode*.

2.2.6 Definition:
Lad δ være en indeksmængde, hvor $\delta \subset \{1, \ldots, n\}$. Den udprikkede kode C_δ består af alle kodeord i koden C med længde n hvor koordinaterne indekseret ved δ fjernes. Dermed bliver længden af C_δ

$$|C_\delta| = n - |\delta|.$$

Nu gives et eksempel på udprikning.

2.2.7 Eksempel:
Betragtes koderne C og C' fra eksempel 2.2.5 og udprikkes indgangen indekseret ved $i = 2$ opnås de to udprikkede koder C_2 og C_2' genereret ved henholdsvis

$$G_2 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad \text{og} \qquad G_2' = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Længden af koderne bliver så $|C_2| = |C_2'| = 6 - 1 = 5$. Disse udprikkede koder benyttes i eksempel 2.2.9, hvor vægtfordelingen beregnes. ◀

Da the support splitting algorithm tager en signatur som input og både invariant og udprikket kode nu er defineret, gives en definition på signatur.

2.2.8 Definition:
En signatur $\mathcal{S}$ over en mængde E afbilder en kode C med længde n og et element i kodens support $i \in \mathcal{J}$ ind i et element i E, således at for alle permutationer σ på $\mathcal{J}$ gælder $\mathcal{S}(C, i) = \mathcal{S}(\sigma(C), \sigma(i))$.

En signatur kan dannes ud fra en invariant afbildning. Lad C og C' være permutationsækvivalente koder, og lad ν være en invariant afbildning, så gælder

$$\nu(C) = \nu(C')$$
$$\{\nu(C_i) \mid i \in 1, \ldots, n\} = \{\nu(C'_i) \mid i \in 1, \ldots, n\},$$

hvor C_i er den udprikkede kode med $\delta = \{i\}$. Dermed haves nu, at hvis de udprikkede koder C_i og C'_i også har samme vægtenumerator, så er koderne C og C' permutationsækvivalente.

2.2.9 Eksempel:
Det undersøges nu, om de to udprikkede koder C_2 og C'_2 fra eksempel 2.2.7 har samme vægtenumerator. Først genereres koderne

$$C_2 = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \end{bmatrix} \right.$$
$$\left. \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \right\}$$

og

$$C'_2 = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \right.$$
$$\left. \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \right\}.$$

Nu laves en tabel over deres vægtfordeling.

	C_2	C'_2
α_0	1	1
α_1	1	2
α_2	2	1
α_3	2	1
α_4	1	2
α_5	1	1

Tabel 7. Vægtfordeling for de udprikkede koder C_2 og C'_2.

Det ses tydeligt, at de to udprikkede koder ikke har samme vægtenumerator, og dermed bekræftes, at koderne C og C' ikke er permutationsækvivalente. ◄

Når der arbejdes med signaturer, benyttes begrebet *diskriminant signatur*.

2.2.10 Definition:
Lad C være en kode med længde n og lad i og j være elementer i supporten $\mathcal{J}$ for koden C. En signatur S er diskriminant for koden C, hvis der eksisterer i og j, således at

$$\mathcal{S}(C, i) \neq \mathcal{S}(C, j).$$

En signatur er fuldt diskriminant, når den er forskellig for hver indgang, altså

$$\mathcal{S}(C, i) \neq \mathcal{S}(C, j) \text{ for alle } i \neq j.$$

Når en signatur er fuldt dikriminant kan permutationen nemt findes. Dette kan beskrives ved at se på koderne C og C'. Hvis permutationen σ af C er $\sigma(C) = C'$, og signaturen er fuldt diskriminant, så haves, at for alle $i \in \{1, \ldots, n\}$ eksisterer der ét unikt j, således at

$$\mathcal{S}(C, i) = \mathcal{S}(C', j).$$

Dette medfører, at σ afbilder i til j, det vil sige $\sigma(i) = j$. The support splitting algorithm skal have to permutationsækvivalente koder samt en fuldt diskriminant signatur for at finde permutationen.

Som tidligere nævnt dannes signaturer ud fra invariante afbildninger. Ofte vælges vægtenumeratoren, men når der arbejdes med store parametre for koderne bliver beregningerne umedgørlige. Beregning af vægtenumeratorene er sværere end beregning af minimumsafstande, der er NP-komplet. Dette motiverer at benytte kodernes *hull*.

2.2.11 Definition:
Lad C være en lineær kode. Kodens hull er fællesmængden mellem koden C og dens duale $C^{\perp}$, hvilket betegnes

$$\mathcal{H}(C) = C \cap C^{\perp}$$

I praksis benyttes vægtenumeratoren for kodens hull istedet, da dette giver en mindre kompleksitet, hvilket behandles i (Sendrier, 2000). I følgende algoritme antages, at en fuldt diskriminant signatur kendes.

Algoritme 8 The support splitting algorithm

Input: To permutationsækvivalente koder C og C' samt en fuldt diskriminant signatur $\mathcal{S}$.
Output: En permutation σ.

1: **for** $i, j \in \mathcal{J}$ **do**
2: **if** $\mathcal{S}(C, i) = \mathcal{S}(C', j)$ **then**
3: $\sigma(i) = j$.
4: **end if**
5: **end for**
6: **return** σ.

Bemærk, at det i algoritme 8 kun er nødvendigt at beregne $2n$ signaturer for at finde permutationen, hvilket har lav kompleksitet. Det er dog antaget, at en fuldt diskriminant signatur er kendt, hvilket ofte ikke er tilfældet. Når der skal findes en fuldt diskriminant signatur, tages udgangspunkt i en signatur, der lidt efter lidt gøres mere og mere diskriminant i håb om, at den til sidst bliver fuldt diskriminant. Dette har en høj kompleksitet. For flere detaljer omkring beregning af en fuldt diskriminant signatur, se (Sendrier, 2000). Nu gives et eksempel på algoritme 8.

2.2.12 Eksempel:

Da dette eksempel har til formål at illustrere brugen af algoritme 8 er det valgt yderst simpelt. Derfor arbejdes der med to ikke-lineære koder over $\mathbb{F}_2$ med længde $n = 4$. De to koder er

$$C = \left\{ \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \right\}$$

og

$$C' = \left\{ \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix} \right\}.$$

Som signatur vælges vægtenumeratoren, og det undersøges, om den er fuldt diskriminant. Dette gøres ved først at undersøge om vægtenumeratoren for C og C' er den samme og derefter for deres udprikkede delkoder.

$$\mathcal{W}_C(x) = \sum_{i=0}^{4} a_i x^i = x^2 + 2x^3$$
$$\mathcal{W}_{C'}(x) = \sum_{i=0}^{4} a_i x^i = x^2 + 2x^3$$

Det ses altså, at de to koder har samme vægtenumerator. Som tidligere nævnt vides, at også de udprikkede koder skal have samme vægtenumerator, når koderne er permutationsækvivalente. Nu dannes de udprikkede koder samt deres vægtenumeratorer for koden C.

$$C_1 = \left\{ \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \right\}, \qquad \mathcal{W}_{C_1}(x) = \sum_{i=0}^{3} a_i x^i = x + x^2 + x^3$$

$$C_2 = \left\{ \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \right\}, \qquad \mathcal{W}_{C_2}(x) = \sum_{i=0}^{3} a_i x^i = x + 2x^2$$

$$C_3 = \left\{ \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \right\}, \qquad \mathcal{W}_{C_3}(x) = \sum_{i=0}^{3} a_i x^i = 2x^2$$

$$C_4 = \left\{ \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \right\}, \qquad \mathcal{W}_{C_4}(x) = \sum_{i=0}^{3} a_i x^i = 2x^2 + x^3$$

Det ses, at signaturen er fuldt diskriminant, da de fire vægtenumeratorer alle er forskellige. For at kunne finde permutationen er det nødvendigt også at kende vægtenumeratorerne

for de udprikkede koder i C'.

$$C_1' = \left\{ \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \right\}, \qquad \mathcal{W}_{C_1}(x) = \sum_{i=0}^{3} a_i x^i = 2x^2$$

$$C_2' = \left\{ \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \right\}, \qquad \mathcal{W}_{C_2}(x) = \sum_{i=0}^{3} a_i x^i = x + x^2 + x^3$$

$$C_3' = \left\{ \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \right\}, \qquad \mathcal{W}_{C_3}(x) = \sum_{i=0}^{3} a_i x^i = 2x^2 + x^3$$

$$C_4' = \left\{ \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \right\}, \qquad \mathcal{W}_{C_4}(x) = \sum_{i=0}^{3} a_i x^i = x + 2x^2$$

Nu benyttes algoritme 8 til at finde permutationen.

$$\begin{aligned} S(C,1) &= S(C',2) & \sigma(1) &= 2 \\ S(C,2) &= S(C',4) & \sigma(2) &= 4 \\ S(C,3) &= S(C',1) & \sigma(3) &= 1 \\ S(C,4) &= S(C',3) & \sigma(4) &= 3. \end{aligned}$$ ◀

Som tidligere nævnt er ovenstående eksempel valgt yderst simpelt, hvilket skyldes, at nøgleangreb ofte er omfattende. Derfor gives der ikke et eksempel på et nøgleangreb i dette projekt, men dog følger her en forklaring på den overordnede teknik baseret på den gennemgåede teori. Når det ønskes at bryde McEliece kryptosystemet ved hjælp af et nøgleangreb kendes den offentlige nøgle $\hat{G}$ og t samt ciffertekstен $\mathbf{y}$. I nogle tilfælde kendes kodens type ikke, hvilket betyder, at det første trin i et nøgleangreb er at gætte på kodens type. Som eksempel antages, at det er koden fra eksempel 1.2.2, der er den rigtige. I appendiks B.9 er den offentlige nøgle genereret. Denne er givet ved $t = 2$ samt

$$\hat{G} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Sammen med den offentlige nøgle kan kodens type også være offentlig. Antages nu, at der er tale om en irreducibel binær Goppa-kode, genereres koden ud fra $\hat{G}$, og de kendte parametre benyttes til at gætte på den rigtige værdi for konstanten m. Koden bliver i dette tilfælde

$$\hat{C} = <\hat{G}> = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \right\}.$$

Det vides, at en Goppa-kode skal opfylde uligheden $k \geq n - m \cdot r \geq 0$, altså $2 \geq 8 - m \cdot 2 \geq 0$, hvilket giver to mulige værdier $m = 3$ eller $m = 4$. Næste trin består i at nedskrive samtlige

polynomier af grad r over $\mathbb{F}_{q^m}$, i dette tilfælde skal der altså nedskrives polynomier af anden grad over enten $\mathbb{F}_{2^3}$ eller $\mathbb{F}_{2^4}$. Dette gøres ikke, da der er henholdsvis $1 \cdot 8 \cdot 8 = 64$ og $1 \cdot 16 \cdot 16 = 256$, i alt 320 mulige polynomier. Bemærk, at et forholdsvis lille eksempel giver over 300 mulige Goppapolynomier, der skal undersøges, hvorfor det er ubetydeligt om kodens type er offentlig. Dette er også grunden til, at Reed-Solomon-koder ikke benyttes, da der kun er den éne kode når parametrene kendes.

Når Goppapolynomierne er fundet er næste trin at vælge ét af dem, generere koden og undersøge, om der findes en signatur, der er fuldt diskriminant for både den fundne kode og koden $\hat{C}$. Er dette tilfældet benyttes support splitting algorithm til at undersøge, om de to koder er ækvivalente. Er det ikke tilfældet, at der findes en fuldt diskriminant signatur, er det, som tidligere nævnt, muligt at forsøge at gøre den fuldt diskriminant eller alternativt at undersøge kodernes hull. Findes der ved disse to metoder ingen ækvivalens, vælges det næste Goppapolynomium, og proceduren gentages. Dette kan ofte være en meget omfattende procedure, og især søgningen efter en fuldt diskriminant signatur er besværlig. Tekninkken bag nøgleangreb ved en Goppa-kode er nu gennemgået, for andre kodetyper er tekninkken lidt anderledes, men dette vil ikke blive gennemgået her.

I dette kapitel er forskellige angreb på McElieces kryptosystem gennemgået. Der er arbejdet med to typer af angreb, meddelelsesangreb og nøgleangreb. I eksemplerne, der er konstrueret meget simple, ses at de algoritmer, der benyttes, har en stor kompleksitet. De valg, der træffes i eksemplerne har vist sig at være yderst gode, og algoritmerne terminerer derfor langt hurtigere end det normalt kan forventes. I næste kapitel arbejdes med quasicykliske koder, da disse koder er gode kandidater til at nedbringe størrelsen af den offentlige nøgle, mens sikkerheden bevares.

KAPITEL 3

Quasicykliske koder

Dette kapitel bygger på (Lally, 2000), (Hall, 2015), (Gaborit, 2004) og (Justesen m.fl., 2004). Når der arbejdes med McEliece kryptosystemet er problemet den store offentlige nøgle. Det ønskes at gøre denne mindre, og den cykliske struktur kan udnyttes til dette. Derfor gennemgås først, i afsnit 3.1, generel teori om cykliske koder, og dette udvides til quasicykliske koder. I afsnit 3.2 omhandlende quasicykliske koder gives forskellige fremgangsmåder til, hvordan en generatormatrix for quasicykliske koder kan reduceres. En anden repræsentation af quasicykliske koder gives i afsnit 3.3, hvor Gröbnerbaser for moduler benyttes. Dette muliggør beregning af dimensionen for quasicykliske koder. Kapitlet afsluttes med en præsentaion af en ny version af McEliece kryptosystemet, der er udviklet af Philippe Gaborit i 2004 (Gaborit, 2004). Han giver et foreslag til, hvordan den offentlige nøglestørrelse kan gøres mindre ved brug af quasicykliske koder, og dette er netop hvad der præsenteres i afsnit 3.4. Gaborit benytter netop en af de gennemgåede metoder til reduktion af generatormatricen. Bemærk, at kodeord i dette kapitel indekseres fra 0, da disse kan betragtes som polynomier, og det dermed letter notationen.

3.1 Cykliske koder

Før der ses på quasicykliske koder betragtes grundlæggende teori om cykliske koder, der er et specialtilfælde af quasicykliske koder. Først defineres, hvad en cyklisk kode er, og derefter gennemgås de egenskaber, der er karakteristiske for denne type.

3.1.1 Definition:

En lineær $[n, k]$-kode $C \subseteq \mathbb{F}_q^n$ kaldes cyklisk, hvis

$$\mathbf{c} = \begin{bmatrix} c_0 & c_1 & \dots & c_{n-1} \end{bmatrix} \in C \Rightarrow \hat{\mathbf{c}} = \begin{bmatrix} c_{n-1} & c_0 & \dots & c_{n-2} \end{bmatrix} \in C.$$

Et *cyklisk skift* vil sige at flytte positionerne i et kodeord én gang mod højre, så den sidste position bliver den første. Dermed ses, at for en cyklisk kode gælder, at ethvert cyklisk skift af et kodeord også giver et kodeord. Der gives nu et eksempel på en cyklisk kode.

3.1.2 Eksempel:
I dette eksempel arbejdes i $\mathbb{F}_2$. Der ses på en $[7,3]$-kode, som har følgende generatormatrix

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Denne generatormatrix danner de følgende $q^k = 2^3 = 8$ kodeord i $\mathbb{F}_2^7$

$$C = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \right.$$
$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$
$$\left. \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \right\}.$$

Det ønskes at se, om der er tale om en cyklisk kode. Hvis det er en cyklisk kode, skal definition 3.1.1 være opfyldt. Det ses hurtigt, at hvis ordet $\begin{bmatrix} c_0 & c_1 & \dots & c_{n-1} \end{bmatrix}$ er indeholdt i C så er $\begin{bmatrix} c_{n-1} & c_0 & \dots & c_{n-2} \end{bmatrix}$ også er et kodeord i C. Dette kan ses på to forskellige måder. Den første måde er ved at tjekke alle kodeordene, men en anden mulighed er at se, om egenskaben gælder for generatormatricen. Hvis det er gældende for generatormatricen, vil det også gælde for resten af kodeordene, da disse er linearkombinationer af kodeordene i generatormatricen. Da generatormatricen G danner alle kodeord, er der tale om en lineær kode. ◄

En anden måde at betragte kodeordene i en cyklisk kode, er ved at se dem som polynomier i $\mathbb{F}_q[x]$. Betragtes et kodeord

$$\mathbf{c} = \begin{bmatrix} c_0 & c_1 & \cdots & c_{n-1} \end{bmatrix} \in C$$

i $\mathbb{F}_q^n$ kan dette repræsenteres ved et polynomium $c(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$ af grad højst $n-1$. Dette svarer til isomorfien

$$\begin{aligned} \mathbb{F}_q^n &\longrightarrow \mathbb{F}_q[x]/\langle x^n - 1 \rangle \\ \begin{bmatrix} c_0 & c_1 & \dots & c_{n-1} \end{bmatrix} &\longmapsto c(x) \mod x^n - 1. \end{aligned}$$

Dette motiverer en alternativ definition af cykliske koder.

3.1.3 Definition:
En cyklisk $[n,k]$-kode C over $\mathbb{F}_q^n$ er et ideal i $\mathbb{F}_q[x]/\langle x^n - 1 \rangle$.

Et kodeord, hvor der er sket ét cyklisk skift vil altså være kodeordet

$$\hat{c}(x) = c_{n-1} + c_0 x + \cdots + c_{n-2}x^{n-1}.$$

Dette ses ved at betragte en omskrivning af $\hat{c}(x)$,

$$\begin{aligned}\hat{c}(x) &= c_{n-1} + c_0 x + \cdots + c_{n-2}x^{n-1} + c_{n-1}x^n - c_{n-1}x^n \\ &= c_0 x + \cdots + c_{n-2}x^{n-1} + c_{n-1}x^n - c_{n-1}x^n + c_{n-1} \\ &= xc(x) - c_{n-1}(x^n - 1) \\ &\equiv xc(x) \mod x^n - 1.\end{aligned}$$

Det cykliske skift svarer dermed til at multiplicere kodeordet $c(x)$ med $x \mod x^n - 1$. Det ses ud fra definitionen af et ideal, at $\hat{c}(x)$ også er et kodeord, da $c(x)$ er indeholdt i idealet C og x er i kvotientringen $\mathbb{F}_q[x]/\langle x^n - 1\rangle$. Bemærk, at det er muligt at foretage flere cykliske skift.

Nu introduceres *generatorpolynomiet* for en cyklisk kode, og dette benyttes til at danne en generatormatrix for koden. Hvis $I = \langle x^n - 1\rangle \subseteq \mathbb{F}_q[x]$ så er $\mathbb{F}_q[x]/I$ en principal ideal ring. Dermed gælder, at enhver kode $C \subseteq \mathbb{F}_q[x]/I$, der er et ideal, kan genereres af ét element fra ringen $\mathbb{F}_q[x]/I$. Altså eksisterer et entydigt monisk polynomium g, der har mindst grad i C, således at $C = \langle g + I\rangle$. Dette g er generatorpolynomiet for den cykliske kode C. En $k \times n$ generatormatrix for C er

$$G = \begin{bmatrix} g_0 & g_1 & \cdots & g_{n-k} & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k} & \cdots & 0 \\ \vdots & & \ddots & \ddots & & \ddots & \vdots \\ 0 & 0 & \cdots & g_0 & g_1 & \cdots & g_{n-k} \end{bmatrix}, \tag{3.1}$$

hvor $g(x) = g_0 + g_1 x + \ldots + g_{n-k}x^{n-k}$.

Nu gives et eksempel, hvor generatorpolynomiet benyttes til at danne en generatormatrix.

3.1.4 Eksempel:

I dette eksempel arbejdes over $\mathbb{F}_2$ med en $[7, 3]$-kode, hvor generatorpolynomiet er $g(x) = x^4 + x^3 + x^2 + 1$. Følgende generatormatrix opnås

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Dette er netop generatormatricen fra eksempel 3.1.2, og g er altså et generatorpolynomium for denne kode. ◀

Dekodning af cykliske koder kan blandt andet gøres ved brug af Meggit dekodning (Huffman m.fl., 2003). Der findes forskellige udgaver af denne type dekodning. Generelt går Meggit dekodning dog ud på, at syndromer med en fejl på position $n - 1$ lagres. Den ene af metoderne går ud på at lave skift af cifferteksten $\mathbf{y}$ indtil, at der er en fejl på position $n - 1$. Denne fejl rettes, hvorefter algoritmen igen laver ét skift og retter den eventuelle fejl i position $n - 2$. Denne teknik fortsættes indtil samtlige positioner er gennemløbet. Meggit

dekodning benytter altså den cykliske egenskab til at rette eventuelle fejl, da ét skift skal være et andet kodeord.

Som tidligere nævnt er en cyklisk kode et specialtilfælde af quasicykliske koder. I næste afsnit gennemgås nogle af egenskaberne for netop quasicykliske koder, da strukturen for disse koder er interessant i forhold til at reducere den offentlige nøglestørrelse i McElice kryptosystemet.

3.2 Teori om quasicykliske koder

I dette afsnit præsenteres den type af koder, der kaldes quasicykliske koder. Der gives en definition og de egenskaber, der er relevante for at strukturen kan benyttes til reduktion af den offentlige nøgle, gennemgås. Derfor vil fokus i afsnittet primært være på generatormatricen for en quasicyklisk kode, da det er reduktion af dennes størrelse, der er interessant.

3.2.1 Definition:
Lad $1 \leq r \leq n$ med $n = rs$, for $s \in \mathbb{N}$. En lineær $[n, k]$-kode $C \subseteq \mathbb{F}_q^n$ siges at være quasicyklisk af orden r, hvis

$$\mathbf{c} = \begin{bmatrix} c_0 & c_1 & \dots & c_{n-1} \end{bmatrix} \in C \Rightarrow \hat{\mathbf{c}} = \begin{bmatrix} c_{n-r} & \dots & c_{n-1} & c_0 & \dots & c_{n-r-1} \end{bmatrix} \in C.$$

Når en quasicyklisk kode er af orden r, siges den at være r quasicyklisk. Det gælder jævnfør definition 3.2.1, at ethvert cyklisk skift med r positioner af et kodeord giver et kodeord. En cyklisk kode svarer til tilfældet, hvor $r = 1$. Bemærk, at hvis C er en cyklisk kode, så er C også en quasicyklisk kode af orden r for ethvert r, der deler n. Nu gives et eksempel på en quasicyklisk kode.

3.2.2 Eksempel:
I dette eksempel arbejdes med en $[6, 2]$-kode over $\mathbb{F}_3$. En generatormatrix for koden er

$$G = \begin{bmatrix} 1 & 2 & 2 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 2 & 2 \end{bmatrix}.$$

Matricen G genererer en kode med $q^k = 3^2 = 9$ kodeord.

$$C = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 2 & 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 1 & 2 & 2 \end{bmatrix} \right.$$
$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 2 & 2 & 2 & 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 \end{bmatrix}$$
$$\left. \begin{bmatrix} 2 & 0 & 0 & 2 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 & 2 & 2 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & 1 & 1 & 0 & 2 & 2 \end{bmatrix} \right\}$$

Det ses tydeligt ud fra generatormatricen, eller ved at betragte kodeordene, at koden er invariant under cykliske skift på 3 positioner, og den er dermed quasicyklisk af orden 3.◀

I eksempel 3.2.2 er generatormatricen valgt ud fra den klassiske forståelse af en generatormatrix, hvor rækkerne danner en basis for koden. Når der arbejdes med

quasicykliske koder, er det muligt at danne en form for generatormatrix, hvor rækkerne er lineært afhængige. Det vil sige, at rækkerne ikke nødvendigvis skal være en basis, men de skal generere hele koden, hvilket i praksis kan være en fordel. Derfor kan en generatormatrix for en quasicyklisk kode over $\mathbb{F}_q^n$ altid findes intuitivt ved brug af cykliske skift af r positioner i ét kodeord s gange, hvor $n = rs$. Kodeordet noteres som

$$\mathbf{c} = \left[\begin{array}{ccccccccccccc} c_1 & c_2 & \dots & c_r & c_{r+1} & c_{r+2} & \dots & c_{2r} & \dots\dots & c_{n-r+1} & c_{n-r+2} & \dots & c_n \end{array} \right] \tag{3.2}$$

og den tilhørende $s \times n$ generatormatrix er følgende.

$$G = \left[\begin{array}{ccccccccccccc} c_1 & c_2 & \dots & c_r & c_{r+1} & c_{r+2} & \dots & c_{2r} & \dots & c_{n-r+1} & c_{n-r+2} & \dots & c_n \\ c_{n-r+1} & c_{n-r+2} & \dots & c_n & c_1 & c_2 & \dots & c_r & \dots & c_{n-2r+1} & c_{n-2r+2} & \dots & c_{n-r} \\ & & & & & & & & & & & & \\ & & & c_{r+2} & \dots & c_{2r} & \dots & c_{3r} & \dots & & c_2 & \dots & c_r \\ & & & c_{r+1} & c_{r+2} & \dots & c_{2r} & c_{2r+1} & \dots & & c_1 & \dots & c_{r-1} \end{array} \right] \tag{3.3}$$

Når en quasicyklisk kode, som i ligning 3.3, er genereret af de cykliske skift af ét kodeord, siges koden at være en *1-generator quasicyklisk kode*. Generelt tales om en v-generator quasicyklisk kode, hvor v er det mindste antal nødvendige kodeord, der skal

til for at generere hele koden ved hjælp af s skift af r positioner. Koden i eksempel 3.2.2 er en 1-generator quasicyklisk kode, hvor $\mathbf{c} = \begin{bmatrix} 1 & 2 & 2 & 0 & 1 & 1 \end{bmatrix}$. Da rækkerne i generatormatricen i eksempel 3.2.2 er lineært uafhængige er $s = k = 2$.

Der gives nu en sætning for, hvordan en generatormatrix, der ikke nødvendigvis har lineært uafhængige rækker, kan konstrueres.

3.2.3 Sætning:
Lad C være en $[n, k]$ quasicyklisk kode af orden r med $n = rs$, så eksisterer der en $k' \times n$ matrix G, hvor $k' \geq k$, der genererer C. Matricen G er på følgende form

$$G = \begin{bmatrix} A_1 & A_2 & A_3 & \dots & A_s \\ A_s & A_1 & A_2 & \dots & A_{s-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_2 & A_3 & A_4 & \dots & A_1 \end{bmatrix}, \tag{3.4}$$

hvor A_i er $\frac{k'}{s} \times r$ matricer på formen

$$A_i = \begin{bmatrix} c^1_{r(i-1)+1} & \dots & c^1_{ri} \\ c^2_{r(i-1)+1} & \dots & c^2_{ri} \\ \vdots & \vdots & \vdots \\ c^{k'/s}_{r(i-1)+1} & \dots & c^{k'/s}_{ri} \end{bmatrix}.$$

Bevis:
Sætningen bevises ved induktion. Der startes med en tom $1 \times n$ matrix G^0, og et tilfældigt kodeord $\mathbf{c} \in C$ betragtes. Opdeles kodeordet i $s = \frac{n}{r}$ dele, hver med r indgange som i ligning (3.2) kan de enkelte dele generelt beskrives som $1 \times r$ matricer på formen

$$A^1_i = \begin{bmatrix} c_{r(i-1)+1} & \dots & c_{ri} \end{bmatrix},$$

hvor $1 \leq i \leq s$. Der ses nu på matricen

$$G^1 = \begin{bmatrix} A^1_1 & A^1_2 & A^1_3 & \dots & A^1_s \\ A^1_s & A^1_1 & A^1_2 & \dots & A^1_{s-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A^1_2 & A^1_3 & A^1_4 & \dots & A^1_1 \end{bmatrix}.$$

Matricen G^1 genererer nu en kode C_1, og det bemærkes, at denne form for matrix svarer til generatormatricen i ligning (3.3). Hvis $C_1 = C$ stoppes, men viser det sig, at den quasicykliske kode, der arbejdes med ikke genereres udelukkende af det valgte kodeord, betragtes nu et kodeord $\mathbf{c}_2$, der ikke genereres af G^1. Dette kodeord deles ligeledes i r lige store dele, og disse tilføjes til A^1_i matricerne, og A^2_i opnås. Det vil sige, at delmatricerne bliver $2 \times r$ på formen

$$A^2_i = \begin{bmatrix} c^1_{r(i-1)+1} & \dots & c^1_{ri} \\ c^2_{r(i-1)+1} & \dots & c^2_{ri} \end{bmatrix},$$

og generatormatricen G^2 opnås som følgende.

$$G^2 = \begin{bmatrix} A_1^2 & A_2^2 & A_3^2 & \dots & A_s^2 \\ A_s^2 & A_1^2 & A_2^2 & \dots & A_{s-1}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_2^2 & A_3^2 & A_4^2 & \dots & A_1^2 \end{bmatrix}$$

Det skal nu tjekkes om denne matrix genererer hele C, hvis dette ikke er tilfældet skal ovenstående procedure fortsættes ved at tjekke, om generatormatircerne $G^3, G^4, \dots, G^j$ genererer hele koden C. Rangen af matricen G^j stiger for hvert j, og den vil stoppe for et bestemt j_0, som opfylder, at $k' = j_0 s$. Da G^{j_0} genrerer hele koden C må $k' \geq k$. Det skal bemærkes, at værdien j_0 kan variere alt efter hvilke kodeord $\mathbf{c}_1, \mathbf{c}_2, \dots$, der betragtes. ■

I praksis arbejdes ofte med generatormatricer på denne form, da det blot er nødvendigt at kende $A_1, \dots, A_s$ for at danne hele generatormatricen. Når der arbejdes med McEliece kryptosystem betyder dette, at størrelsen på $\hat{G}$ i den offentlige nøgle reduceres fra en $k \times (n-k)$ generatormatrix på systematisk form til en $\frac{k'}{s} \times n$ matrix. I praksis vælges k' tæt på k, så delmatricerne A_i får så lille en dimension som muligt. Dette er også idéen bag Gaborits version, der præsenteres i afsnit 3.4.

3.2.4 Eksempel:

En generatormatrix for den quasicykliske $[6, 2]$-kode over $\mathbb{F}_3$ i eksempel 3.2.2 dannes nu. Der ses først på kodeordet $\mathbf{c}_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$. Det vides, at $r = 3$ og $s = 2$, demed bliver $A_1^1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ og $A_2^1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$. Generatormatricen skal nu dannes ud fra disse to så

$$G^1 = \begin{bmatrix} A_1^1 & A_2^1 \\ A_2^1 & A_1^1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Det kan tydeligt ses, at denne generatormatrix ikke danner hele koden, og derfor skal et kodeord, der ikke genereres at den nuværende generatormatrix G^1 tilføjes til A_i^1. Kodeordet $\mathbf{c}_2 = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 \end{bmatrix}$ tilføjes så

$$A_1^2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad A_2^2 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 2 & 2 \end{bmatrix}$$

og

$$G^2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 2 & 2 & 2 & 1 & 1 & 1 \end{bmatrix}.$$

Da G^2 genererer hele koden er en generatormatrix for koden altså $G = G^2$. Det er i McEliece kryptosystem dermed kun nødvendigt at sende A_1 og A_2 i den offentlige nøgle.

Valget af $\mathbf{c}_1$ og $\mathbf{c}_2$ gør, at størrelsen på den offentlige nøgle ikke ændres i forhold til generatormatricen fra eksempel 3.2.2.

Bemærk dog, at var valget af det første kodeord faldet på $\mathbf{c}_1 = \begin{bmatrix}1 & 2 & 2 & 0 & 1 & 1\end{bmatrix}$, ville delmatricerne være $A_1^1 = \begin{bmatrix}1 & 2 & 2\end{bmatrix}$ og $A_2^1 = \begin{bmatrix}0 & 1 & 1\end{bmatrix}$, hvormed generatormatricen vil blive

$$G^1 = \begin{bmatrix}1 & 2 & 2 & 0 & 1 & 1\\ 0 & 1 & 1 & 1 & 2 & 2\end{bmatrix}.$$

Denne generatormatrix genererer hele koden, og derfor er der tale om en 1-generator, og det er derfor kun nødvendigt at kende A_1 og A_2, til at danne generatormatricen. Den offentlige nøgle i McEliece kryptosystemet kan dermed reduceres til en 1×6 matrix, da dette svarer til den 1-generator, der er for koden. ◄

En anden egenskab, der følger af definitionen på quasicykliske koder er, at de er invariante under permutationen

$$\pi : \quad i \quad \longmapsto \quad i + tr \mod n,$$

hvor $i = 0, 1, \ldots, n-1$ angiver indekseringen af en position i $\mathbf{c}$ og $t \in \mathbb{Z}$.

3.2.5 Eksempel:
Betragt koden fra eksempel 3.2.2, der har længde $n = 6$ og $r = 3$. Der er kun to mulige permutationer, som koden er invariant overfor, identiteten og skift på tre positioner, hvilket sker når t er henholdsvis lige og ulige. ◄

Ved permutation af rækkerne i en generatormatrix sker der ingen ændring af koden, der genereres, og både r og s forbliver de samme. Ved søjlepermutation ændres koden der genereres af rækkerne, men da det kun er positionerne i kodeordene, der ændres, er den kode, der opnås, ækvivalent med den oprindelige, da den har samme længde, dimension og vægtfordeling.

3.2.6 Eksempel:
Betragt igen koden fra eksempel 3.2.2 med generatormatricen

$$G = \begin{bmatrix}1 & 2 & 2 & 0 & 1 & 1\\ 0 & 1 & 1 & 1 & 2 & 2\end{bmatrix},$$

hvor $r = 3$ og $s = 2$, så kan en ækvivalent kode findes ved permutationen ν, der er givet ved $\nu(1) = 1$, $\nu(2) = 5$, $\nu(3) = 3$, $\nu(4) = 2$, $\nu(5) = 4$ og $\nu(6) = 6$. Dermed opnås generatormatricen

$$G = \begin{bmatrix}1 & 0 & 2 & 1 & 2 & 1\\ 0 & 1 & 1 & 2 & 1 & 2\end{bmatrix},$$

og det kan ses, at G er på systematisk form. Det ses yderligere, at søjle 1 og 2 danner en cyklisk delmatrix, og tilsvarende gælder for søjle 3 og 4 samt 5 og 6. Dermed er generatormatricen omskrevet til at bestå af tre cykliske delmatricer. ◄

Som det illustreres i eksempel 3.2.6 kan generatormatricerne for quasicykliske koder omskrives, så de består af *cykliske delmatricer*. Dette svarer til at generatormatricen i ligning (3.4) består af cykliske matricer på formen

$$A_{i,j} = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{s-1} \\ a_{s-1} & a_0 & a_1 & \cdots & a_{s-2} \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ a_1 & a_2 & a_3 & \cdots & a_0 \end{bmatrix}.$$

Fremadrettet betegner $A_{i,j}$ cykliske $s \times s$ matricer. Generatormatricen i ligning (3.4) kan dermed omskrives til at bestå af cykliske delmatricer. Metoden til dette er at samle søjler indekseret ved $i, r+i, 2r+i, \ldots$ for $i = 1, 2, \ldots, s$, så der dannes r blokke med længde s. Derefter skal rækkerne grupperes sådan, at de rækker indekseret ved $j, v+j, 2v+j, \ldots$ for $j = 1, 2, \ldots, s$ samles. Dette betyder, at rækkerne dannet af 1-generator kodeordet samles på de første s rækker, rækkerne dannet af 2-generator kodeordet samles på de næste s rækker og så videre. Der opnåes altså en $sv \times sr = k' \times n$ generatormatrix på formen

$$G = \begin{bmatrix} A_{1,1} & A_{1,2} & \ldots & A_{1,r} \\ A_{2,1} & A_{2,2} & \ldots & A_{2,r} \\ \vdots & \vdots & \vdots & \vdots \\ A_{v,1} & A_{v,2} & \ldots & A_{v,r} \end{bmatrix},$$

hvor rækkerne ikke nødvendigvis er lineært uafhængige. Når rækkerne i G er lineært uafhængige vil permutationerne af henholdsvis søjler og rækker danne en generatormatrix på systematisk form, hvilket også er tilfældet i eksempel 3.2.6.

$$G_s = \begin{bmatrix} I_s & 0 & \ldots & \ldots & 0 & A_{1,v+1} & A_{1,v+2} & \ldots & A_{1,r} \\ 0 & I_s & 0 & \ldots & 0 & A_{2,v+1} & A_{2,v+2} & \ldots & A_{2,r} \\ \vdots & 0 & \ddots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & & \ddots & 0 & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & I_s & A_{v,v+1} & A_{v,v+2} & \ldots & A_{v,r} \end{bmatrix},$$

hvor I_s betegner en $s \times s$ identitetsmatrix, og $A_{i,v+i}$ er cykliske $s \times s$ matricer. Når en systematisk generatormatrix haves, er første del af matricen en $sv \times sv$ identitetsmatrix. Dermed vil det kun være nødvendigt at sende den ikke trivielle del af generatormatricen som den offentlige nøgle i McEliece kryptosystemet, dette vil være en $sv \times s(r-v)$ matrix, hvormed den offentlige nøglestørrelse vil være reduceret.

3.2.7 Eksempel:

I dette eksempel arbejdes med en $[14, 4]$ systematisk quasicyklisk kode over $\mathbb{F}_2$, der altså består af de 16 kodeord, der ses i appendiks B.10. Det ses, at $r = 7$ og $s = 2$. Kodeordet

$$\mathbf{c}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

vælges som første generator og G^1 dannes.

$$A_1^1 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \qquad A_2^1 = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

og

$$G^1 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Det ses, at G^1 ikke genererer hele koden, men kun 4 kodeord. Derfor vælges endnu et kodeord, og G^2 dannes. Kodeordet

$$\mathbf{c}_2 = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

vælges som anden generator, og A_1^2 samt A_2^2 dannes som følgende.

$$A_1^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad A_2^2 = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix},$$

hvormed

$$G^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Det ses, at G^2 genererer hele koden og dermed $G = G^2$, $A_1 = A_1^2$, $A_2 = A_2^2$ samt $v = 2$. Desuden ses, at A_i er 2×7 matricer. Det er altså kun nødvendigt at sende en 2×14 matrix ved McEliece kryptosystem. Nu sættes generatormatricen G på systematisk form ved hjælp af søjle- og rækkepermutation. Først permuteres søjlerne i G så søjle 1 og 8 samles, 2 og 9 samles og så videre.

$$G_s = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Rækkerne permuteres så række 1 og 3 samles samt 2 og 4.

$$G_s = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Det ses nu, at G_s er på systematisk form, og at de nye A_i er cykliske. ◄

Teorien om quasicykliske koder, og herunder hvordan generatormatricen kan dannes af delmatricer, er nu gennemgået. I næste afsnit præsenteres en anden måde at se på quasicykliske koder er ud fra teorien om moduler.

3.3 Quasicykliske koder ved Gröbnerbaser

Det antages i dette afsnit, at den grundlæggende teori om moduler er kendt, og den kan ses i (Adams m.fl., 2000) og (Cox m.fl., 2005). Quasicykliske koder kan karakteriseres ved brug af teorien om moduler og undermoduler. Den quasicykliske kode C med længde $n = sr$ kan betragtes som et $\frac{R}{I}$-undermodul af modulet $\left(\frac{R}{I}\right)^r$, hvor $R = \mathbb{F}_q[x]$ og $I = \langle x^s - 1\rangle$. Kodeordene i C er på formen

$$\mathbf{c} = \left[\begin{array}{cccccccccc} c_{0,0} & \dots & c_{0,r-1} & c_{1,0} & \dots & c_{1,r-1} & \dots\dots & c_{s-1,0} & \dots & c_{s-1,r-1} \end{array} \right],$$

og de kan opfattes som polynomiumsvektorer

$$\mathbf{c} = \left[\begin{array}{cccc} c_0 + I & c_1 + I & \dots & c_{r-1} + I \end{array} \right] \in \left(\frac{R}{I}\right)^r,$$

hvor c_i betegner polynomiet $c_i(x) = c_{0,i} + c_{1,i}x + \cdots + c_{s-1,i}x^{s-1}$. Bemærk, at $\deg(c_i) < s$ for $i = 1, 2, \dots, r-1$. Fremadrettet vil sideklassenotationen ikke fremgå, og kodeord noteres som følgende vektor

$$\mathbf{c} = \left[\begin{array}{cccc} c_0 & c_1 & \dots & c_{r-1} \end{array} \right]. \tag{3.5}$$

Det ses som tidligere, at quasicykliske koder er invariante under skift med tr positioner, hvor $t \in \mathbb{Z}$. Dette svarer til at permutere c_i'erne i (3.5), hvilet betyder at de r dele bibeholder rækkefølgen, men ikke hvilken c_i, der står først. En en både at bibeholde den quasicykliske struktur er ved at betragte hvert c_i og lave et cyklisk skift med 1 position (Berger m.fl., 2009). Denne definition svarer til, at hvert c_i er en form for cyklisk kode, sammensætningen af disse giver en ækvivalent quasicyklisk struktur. Nu begynder forarbejdet til at danne en Gröbnerbasis for en quasicyklisk kode. Idéen er at vise en sammenhæng mellem to undermoduler, og ud fra disse vise at hvis en bestemt monomiel ordning benyttes, så kan en minimal Gröbnerbasis dannes. Ud fra denne kan kodens dimension beregnes. Første skridt er at en en-til-en sammenhæng mellem to undermoduler derfor ses på følgende. Der eksisterer en surjektiv homomorfi, som ved brug af sideklassenotation beskrives ved

$$\begin{array}{rccc} \tau: & R^r & \longrightarrow & \left(\frac{R}{I}\right)^r \\ & \mathbf{f} = (f_1, f_2, \dots, f_r) & \longmapsto & (f_1 + I, f_2 + I, \dots, f_r + I) \end{array}$$

mellem polynomiumsvektorerne i R^r og $\left(\frac{R}{I}\right)^r$. Nulrummet for afbildningen τ er

$$\widetilde{\mathcal{K}} = \left\{ \mathbf{f} \in R^r \mid \tau(\mathbf{f}) = \mathbf{0} \in \left(\frac{R}{I}\right)^r \right\},$$

der er et undermodul i det frie modul R^r. Bemærk, at nulvektoren i $\left(\frac{R}{I}\right)^r$ svarer til $(I, I, \dots, I)$ i R^r. Dermed kan nulrummet også noteres ved

$$\widetilde{\mathcal{K}} = \{\mathbf{f} = (f_1, f_2, \dots, f_r) \in R^r \mid f_i = k(x^s - 1) \text{ for } k \in \mathbb{F}_q[x] \text{ og } 1 \leq i \leq r\},$$

og det ses altså, at $\widetilde{\mathcal{K}}$ er genereret af

$$\begin{aligned}\widetilde{\kappa} &= \{(x^s-1,0,\ldots,0),(0,x^s-1,0,\ldots,0),\ldots,(0,\ldots,0,x^s-1)\}\\ &= \{(x^s-1)\mathbf{e}_i \text{ for } i=1,\ldots,r\} \subset R^r.\end{aligned}$$

Nu vises, at der eksisterer en en-til-en sammenhæng mellem undermodulet C i $\left(\frac{R}{I}\right)^r$ og undermodulet $\widetilde{C}$ i $\frac{R^r}{\widetilde{\mathcal{K}}}$. Afbildningen

$$\begin{array}{rccc}\theta: & \dfrac{R^r}{\widetilde{\mathcal{K}}} & \longrightarrow & \left(\dfrac{R}{I}\right)^r \\ & (f_1,f_2,\ldots,f_r)+\widetilde{\mathcal{K}} & \longmapsto & (f_1+I,f_2+I,\ldots,f_r+I),\end{array}$$

definerer isomorfien

$$\frac{R^r}{\widetilde{\mathcal{K}}} \cong \left(\frac{R}{I}\right)^r,$$

da $\widetilde{\mathcal{K}}$ er nulrummet for afbildningen τ, der har $\left(\frac{R}{I}\right)^r$ som billede. Altså eksisterer en en-til-en sammenhæng mellem C og $\widetilde{C}$. Dette benyttes senere til at bestemme dimensionen af koden C, men før dette kan gøres, er det nødvendigt at finde en Gröbnerbases for $\widetilde{C}$. En v-generator kode C, der er genereret af elementerne $\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_v \in \left(\frac{R}{I}\right)^r$, er et undermodul $C=\langle \mathbf{a}_1,\mathbf{a}_2,\ldots \mathbf{a}_v\rangle$ i $\left(\frac{R}{I}\right)^r$. Det tilsvarende undermodul $\widetilde{C}$ i R^r er genereret af $\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_v \in R^r$ og elementerne i $\widetilde{\mathcal{K}}$, altså

$$\widetilde{C} = \langle \mathbf{a}_1,\mathbf{a}_2,\ldots \mathbf{a}_v,(x^s-1)\mathbf{e}_1,(x^s-1)\mathbf{e}_2,\ldots,(x^s-1)\mathbf{e}_r\rangle.$$

Nu haves en v-generator kode $\widetilde{C}$, der er et undermodul til det frie modul R^r, og dermed kan teorien om Gröbnerbaser benyttes. I den forbindelse skal en monomiel ordning benyttes, og i dette projekt er POT-ordningen med $\mathbf{e}_1 > \mathbf{e}_2 > \cdots > \mathbf{e}_r$ valgt. I resten af dette kapitel bliver der arbejdet med POT-ordningen, og den efterfølgende teori er kun gældende for netop denne ordning. Med POT-ordningen haves, at for to monomier

$$\begin{aligned}\mathbf{M}_1 &= x^\alpha \mathbf{e}_i\\ \mathbf{M}_2 &= x^\beta \mathbf{e}_j\end{aligned}$$

i R^r gælder, at $\mathbf{M}_1 < \mathbf{M}_2$ hvis og kun hvis enten $i>j$ eller $i=j$ og $\alpha<\beta$.

Det haves, at $0 \subseteq \widetilde{\mathcal{K}} \subseteq \widetilde{C} \subseteq R^r$. Lad $\widetilde{\mathcal{G}}$ være en minimal Gröbnerbasis sådan, at $\widetilde{C} = \left\langle \widetilde{\mathcal{G}} \right\rangle$. Da $\widetilde{\mathcal{K}}$ indeholder vektorerne $(x^s-1)\mathbf{e}_i$ for $i=1,2,\ldots,r$, så har $\widetilde{\mathcal{K}}$ det ledende monomium $\mathrm{lm}((x^s-1)\mathbf{e}_i)$ på i'te position.

Ud fra teorien om Gröbnerbaser vides, at der eksisterer et $\mathbf{g} \in \widetilde{\mathcal{G}}$ således, at $\mathrm{lm}(\mathbf{g})$ deler $\mathrm{lm}((x^s-1)\mathbf{e}_i)$ for $i=1,2,\ldots,r$. Dermed må $\mathrm{lm}(\mathbf{g})$ være på den i'te position. Der eksisterer altså et $\mathbf{g} \in \widetilde{\mathcal{G}}$ for hvert $i=1,2,\ldots,r$, hvorom det gælder, at $\mathrm{lm}(\mathbf{g}) = X\mathbf{e}_i$, hvor $X \in R$.

Lad $\mathbf{g}_1$ og $\mathbf{g}_2$ være to elementer i $\widetilde{\mathcal{G}}$, der opfylder ovenstående og har ledende monomium på samme position. Antag, at

$$\begin{aligned}\mathrm{lm}(\mathbf{g}_1) &= x^\alpha \mathbf{e}_i\\ \mathrm{lm}(\mathbf{g}_2) &= x^\beta \mathbf{e}_i\end{aligned}$$

for et i, hvor $1 \leq i \leq r$. Da der kun arbejdes med én variabel x gælder, at ét af de ledende monomier for $\mathbf{g}_1$ og $\mathbf{g}_2$ vil dele det andet. Dermed haves, at hvis $\widetilde{\mathcal{G}}$ er en minimal Gröbnerbasis, så vil den bestå af nøjagtigt r elementer $\mathbf{g}_i$ med ledende monomier på forskellige positioner. En sådan Gröbnerbasis kan noteres ved

$$\widetilde{\mathcal{G}} = \{\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_r\},$$

og det kan uden tab af generalitet antages, at

$$\mathrm{lm}(\mathbf{g}_i) = x_i^{\alpha}\mathbf{e}_i \text{ for } i = 1, 2, \ldots, r,$$

hvor $x_i^{\alpha} \in R$. Da der arbejdes med POT-ordningen og en minimal Gröbnerbasis følger, at for $\mathbf{g}_i$, der har ledende monomium på i'te position, har enhver position j, hvor $1 \leq j < i$, værdien nul. Dermed må $\mathbf{g}_i$ være på formen

$$\mathbf{g}_i = \left(0, 0, \ldots, 0, g_i^i, g_i^{i+1}, \ldots, g_i^r\right) \in R^r, \tag{3.6}$$

hvor $g_i^i \neq 0$. Heraf følger, at elementerne i $\widetilde{\mathcal{G}}$ er

$$\begin{aligned}
\mathbf{g}_1 &= \begin{bmatrix} g_1^1 & g_1^2 & g_1^3 & \ldots & g_1^r \end{bmatrix} \\
\mathbf{g}_2 &= \begin{bmatrix} 0 & g_2^2 & g_2^3 & \ldots & g_2^r \end{bmatrix} \\
\mathbf{g}_3 &= \begin{bmatrix} 0 & 0 & g_3^3 & \ldots & g_3^r \end{bmatrix} \\
&\vdots \\
\mathbf{g}_r &= \begin{bmatrix} 0 & 0 & \ldots & 0 & g_r^r \end{bmatrix}.
\end{aligned}$$

De moniske polynomier $g_i^i \neq 0$, hvor $i = 1, 2, \ldots, r$, kaldes *diagonal elementer* i $\widetilde{\mathcal{G}}$. Det gælder altså, at g_i^i indeholder det ledende monomium for $\mathbf{g}_i$.

Bemærk, at diagonalelementerne i Gröbnerbasen $\mathcal{G}$ er en direkte generalisering af generatorpolynomiet for en cyklisk kode, se ligning (3.1). Nu gives en sætning, der giver en egenskab for disse diagonalelementer.

3.3.1 Sætning:
Diagonalelementet g_i^i deler $x^s - 1$ for ethvert i, hvor $1 \leq i \leq r$.

Bevis:
Det vides, at for ethvert i, hvor $1 \leq i \leq r$, gælder, at $(x^s - 1)\mathbf{e}_i \in \widetilde{\mathcal{K}}$ og dermed at $(x^s - 1)\mathbf{e}_i \in \widetilde{C}$, da $\widetilde{\mathcal{K}} \subset \widetilde{C}$. Desuden vides, som tidligere nævnt, at der eksisterer et $\mathbf{g} \in \widetilde{\mathcal{G}}$ således, at $\mathrm{lm}(\mathbf{g})$ deler $\mathrm{lm}((x^s - 1)\mathbf{e}_i) = x^s - 1$ for $i = 1, 2, \ldots, r$. Altså er $\mathrm{lm}(\mathbf{g})$ på den i'te position, og det er altså elementet $\mathbf{g}_i$ i Gröbnerbasen, der skal tages i betragtning. Dette følger, da der arbejdes med en minimal Gröbnerbasis og POT-ordningen med $\mathbf{e}_1 > \mathbf{e}_2 > \cdots > \mathbf{e}_r$ og det vides, at der er præcis r elementer i Gröbnerbasen, der opfylder at $\mathbf{g}_1 > \mathbf{g}_2 > \cdots > \mathbf{g}_r$. Ud fra ligning (3.6) må $\mathrm{lm}(\mathbf{g}_i)$, der er på i'te position, være elementet g_i^i. Dermed følger at g_i^i deler $x^s - 1$. ■

Nu vides altså at $\widetilde{\mathcal{G}} = \{\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_r\}$ er en minimal Gröbnerbasis for $\widetilde{C}$. Desuden vides at der eksisterer en en-til-en sammenhæng mellem koden C, der er et undermodul til $\left(\frac{R}{I}\right)^r$, og undermodulet $\widetilde{C}$. Dimensionen af koden C er givet ved

$$k = sr - \sum_{i=1}^{r} \deg(g_i^i), \tag{3.7}$$

hvor (g_i^i) er diagonalelementer i Gröbnerbasen $\widetilde{\mathcal{G}}$. Der gives nu et eksempel på en quasicyklisk kode, der genereres ud fra ovenstående teori.

3.3.2 Eksempel:

Der arbejdes med en 3-generator kode C, der genereres af $\langle \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3 \rangle$ over $\mathbb{F}_3$. Koden er cyklisk med skift på $r = 4$ positioner, og længden af kodeordene er 20, hvormed $s = 5$. Det ønskes nu at danne en Gröbnerbasis for undermodulet $\widetilde{C}$, der som tidligere nævnt genereres som

$$\widetilde{C} = \langle \mathbf{a}_1, \mathbf{a}_2, \ldots \mathbf{a}_k, (x^s - 1)\mathbf{e}_1, (x^s - 1)\mathbf{e}_2, \ldots, (x^s - 1)\mathbf{e}_r \rangle .$$

Da der er en en-til-en sammenhæng mellem C og $\widetilde{C}$ gælder som tidligere nævnt at det ud fra en Gröbnerbasis til $\widetilde{C}$ er muligt at finde dimensionen for koden C. I dette eksempel arbejdes med følgende elementer haves.

$$\mathbf{a}_1 = \begin{bmatrix} x^4+1 \\ x^3+x^2+1 \\ 2x^4 \\ x^3+2x \end{bmatrix} \quad \mathbf{a}_2 = \begin{bmatrix} x^2+2x \\ x^4+x^3+x+1 \\ x^2+2x \\ x^3+x^2+2 \end{bmatrix} \quad \mathbf{a}_3 = \begin{bmatrix} x^2+x+1 \\ 2x^3+x \\ x^3+2x^2+x+2 \\ x^4+x \end{bmatrix}.$$

Den reducerede Gröbnerbasis, der altså også er minimal, dannes i SAGE. Beregningen af denne kan ses i appendiks B.14, og giver følgende $\mathbf{g}_i$ for $i = 1, \ldots, 4$

$$\mathbf{g}_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ -x^2+1 \end{bmatrix} \quad \mathbf{g}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -x^3+x+1 \end{bmatrix}$$

$$\mathbf{g}_3 = \begin{bmatrix} 0 \\ 0 \\ x-1 \\ x-1 \end{bmatrix} \quad \mathbf{g}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ x^4+x^3+x^2+x+1 \end{bmatrix}$$

Herefter beregnes dimensionen af den quasicykliske kode C ved brug af igning (3.7). Da Gröbnerbasen for $\widetilde{C}$ netop er dannet kan dimensionen af C let beregnes, og følgende haves.

$$\begin{aligned} k &= 5 \cdot 4 - \sum_{i=1}^{4} \deg(g_i^i) \\ &= 5 \cdot 4 - \big(\deg(1) + \deg(1) + \deg(x-1) + \deg(x^4+x^3+x^2+x+1)\big) \end{aligned}$$

Dermed bliver dimensionen af denne quasicykliske kode $k = 20 - (1+4) = 15$. ◀

I dette projekt arbejdes med McEliece kryptosystem og i næste afsnit gives en version af dette, hvor quasicykliske koder benyttes. Denne version er udviklet af Gaborit, og hans idé er at benytte netop den quasicykliske egenskab til at reducere den offentlige nøglestørrelse.

3.4 McEliece kryptosystemet med quasicykliske koder

Problemet med McEliece kryptosystemet er som tidlige nævnt den store offentlige nøgle. Philippe Gaborit giver i artiklen *Shorter keys for code based cryptography* et foreslag til en ændring af McEliece kryptosystemet, hvor den offentlige nøglestørrelse kan reduceres ved hjælp af quasicykliske koder. Han udvikler i den forbindelse en ny version af kryptosystemet, og netop denne gennemgås i dette afsnit.

Idéen bag foreslaget er, at benytte den quasicykliske kode, og den egenskab, der betyder, at en generatormatrix kan dannes, så det kun er nødvendigt at sende delmatricerne $A_1, A_2, \dots A_s$, hvis C være en quasicyklisk kode af orden r. Det er dermed muligt, at koden C kan genereres af generatormatricen på formen som i sætning 3.2.3. Denne generatormatrix er netop dannet af de s delmatricer $A_1, A_2, \dots, A_s$, der alle har størrelse $\frac{k'}{s} \times r$.

Den offentlige nøgle genereres på en anden måde end ved McEliece kryptosystemet, hvor både en permutationsmatrix P og en invertibel matrix S skal benyttes. Her skal der i stedet kun benyttes en permutationsmatrix Π. Det ønskes nu at danne denne permutation, som skal være på r koordinater, der permuterer s sæt af r koordinater. Dette vil sige $(1, \dots, r), (r+1, \dots, 2r), \dots, (n-r+1, \dots, n)$, og denne permutation noteres med π. Generatormatricen G fra ligning (3.4) ændres med denne permutation til matricen G', og da der permuteres r koordinater vil søjlerne i A_i for $i = 1, 2, \dots, s$ stadig være samlet. Det betyder, at permutationen π på A_i bliver

$$A_1\pi, A_2\pi, \dots, A_s\pi.$$

Der kan nu defineres en permutationsmatrix med permutationen π, der virker på de s sæt af r koordinater. Denne matrix noteres Π, og er på følgende form.

$$\Pi = \begin{bmatrix} \pi & 0 & \dots & 0 \\ 0 & \pi & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \pi \end{bmatrix} \tag{3.8}$$

Matricen G' kan derved dannes

$$\begin{aligned} G' &= G\Pi \\ &= \begin{bmatrix} A_1 & A_2 & A_3 & \dots & A_s \\ A_s & A_1 & A_2 & \dots & A_{s-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_2 & A_3 & A_4 & \dots & A_1 \end{bmatrix} \cdot \begin{bmatrix} \pi & 0 & \dots & 0 \\ 0 & \pi & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \pi \end{bmatrix} \\ &= \begin{bmatrix} A_1\pi & A_2\pi & A_3\pi & \dots & A_s\pi \\ A_s\pi & A_1\pi & A_2\pi & \dots & A_{s-1}\pi \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_2\pi & A_3\pi & A_4\pi & \dots & A_1\pi \end{bmatrix}. \end{aligned}$$

Det ses, at G' er dannet ud fra samme princip som G, og det er ud fra dette Gaborit foreslår, at den offentlige nøgle kan reduceres til at indeholde matricerne $A_i\pi$ for $i = 1, 2, \dots, s$, i stedet for hele generatormatricen, hvor rækkerne er lineært uafhængige. Disse matricer er som tidligere nævnt $\frac{k'}{s} \times r$ matricer, hvilket betyder at den offentlige nøgle med denne fremgangsmåde vil have en størrelse på $\frac{k'}{s}n$ i stedet for en nøglestørrelse på kn, da k' vælges tæt på k må dette reducere nøglestørrelsen. Nu gives en algoritme til, hvordan den offentlige og private nøgle genereres ved Gaborits version.

Algoritme 9 Nøglegenerering i Gaborits version

1: Vælg en r-quasicyklisk $[n, k, d]$-kode således at $W_{q,n,d,t} \geq 2^\kappa$.
2: Bestem en tilfældig generatormatrix G for C på formen i sætning 3.2.3.
3: Bestem en permutationsmatrix Π som i ligning (3.8).
4: Beregn $G' = G\Pi$.
5: **return** Den offentlige nøgle (matricerne $(A_1\pi, A_2\pi, \dots, A_s\pi), r, t$) og den private nøgle ($\Pi$, dekodningsalgoritme for C).

Der gives nu et eksempel på denne nøglegenerering.

3.4.1 Eksempel:

Bob ønsker at danne en offentlig nøgle til McEliece kryptosystemet i Gaborits version, hvor det kun er delmatricerne fra sætning 3.2.3, altså A_i for $i = 1, 2, \dots, s$, der sendes. Der arbejdes med en $[10, 3, 4]$-kode over $\mathbb{F}_3$ med følgende generatormatrix

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 & 1 \end{bmatrix}.$$

Det er muligt at se alle kodeord samt beregning af minimumsafstand fra SAGE i appendiks B.11. Det ses ud fra G, at $r = 5$, og dermed at $s = 2$. Nu danner Bob en generatormatrix på formen fra sætning 3.2.3. Først vælger han $\mathbf{c}_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$, og dermed haves

$$A_1^1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad \text{samt} \qquad A_2^1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Nu dannes

$$G^1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Da G^1 ikke genererer hele koden, vælger han et kodeord

$$\mathbf{c}_2 = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 & 1 \end{bmatrix}$$

og matricerne

$$A_1^2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 2 & 1 \end{bmatrix} \qquad \text{samt} \qquad A_2^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 \end{bmatrix}$$

dannes. Nu bliver

$$G^2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 & 1 \end{bmatrix},$$

og det ses, at G^2 genererer hele koden, hvormed $G = G^2$. Nu skal Bob finde en permutation, der permuterer 5 søjler. Han vælger følgende

$$\pi(i) = (i+1) \mod 5,$$

for $i = 1, 2, 3, 4, 5$, og dermed får han følgende permutationsmatrix

$$\Pi = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Nu beregnes

$$G' = G\Pi = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 \end{bmatrix}.$$

Dermed opnås

$$A_1\pi = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 2 \end{bmatrix} \qquad \text{og} \qquad A_2\pi = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 2 \end{bmatrix},$$

og Bob sender $(A_1\pi, A_2\pi, r = 5, t = 1)$ som den offentlige nøgle. ◀

Når der skal indkodes en meddelelse med den offentlige nøgle er det nødvendigt, at denne generatormatrix har k lineært uafhængige rækker. Den matrix G', der dannes af delmatricerne $A_1\pi, A_2\pi, \ldots, A_s\pi$ har ud fra konstruktion af denne ikke nødvendigvis k rækker, men kan have flere. Det er derfor nødvendigt at danne en ny generatormatrix M ud fra $A_1\pi, A_2\pi, \ldots, A_s\pi$, som kun består af k lineært uafhængige rækker.

Matricen M dannes ved først at se på det kodeord i G, der er dannet ud fra den første række i $A_1\pi, A_2\pi, \ldots, A_s\pi$, hvilket noteres $\mathbf{c}_1$. Dette kodeord tilføjes nu til matricen M_1, som kun indeholder $\mathbf{c}_1$. Herefter laves der et cyklisk skift af $\mathbf{c}_1$ på r koordinater, dette nye kodeord noteres $\mathbf{c}_1'$. Der skal nu ses på om kodeordet $\mathbf{c}_1'$ genereres af M_1, hvis dette ikke er tilfældet dannes M_2, som M_1 hvor $\mathbf{c}_1'$ tilføjes. Hvis $\mathbf{c}_1'$ genereres dannes $M_2 = M_1$.

Herefter skal fremgangsmåden gentages $s-1$ gange, ved at lave $r \cdot i$ skrift af koordinaterne i $\mathbf{c}_1$, hvor $i = 1, 2, \ldots s$. Hvis kodeordene, der dannes ved disse skift ikke genereres af M_i tilføjes disse hertil indtil M_s er dannet. Herefter ses på $\mathbf{c}_2$, som er anden række i $A_1\pi, A_2\pi, \ldots, A_s\pi$, og dette tilføjes M_s, som dermed noteres M_{s+1}.

Der skal nu benyttes samme fremgangsmåde, hvor der laves cykliske skift af $\mathbf{c}_2$, som det er gjort for $\mathbf{c}_1$. Dette gøres for alle rækkerne i $A_1\pi, A_2\pi, \ldots, A_s\pi$, som der er $\frac{k'}{s}$ af, og resultatet af dette er en generatormatrix $M_{k'} = M$ med k rækker, og det er derfor nu muligt at indkode meddelelsen med denne generatormatrix. En algoritme til dette præsenteres nu.

Algoritme 10 Indkodning i Gaborits version

Input: En meddelelse $\mathbf{m} \in \mathbb{F}_q^k$, den offentlige nøgle, matricerne $A_1\pi, A_2\pi, \ldots, A_s\pi$, den quasicyklike orden r samt parameteren t.

Output: En cifferskrift $\mathbf{y} \in \mathbb{F}_q^n$.

1: Dan generatormatricen M som beskrevet ovenfor ud fra $A_1\pi, A_2\pi, \ldots, A_s\pi$.
2: Beregn $\mathbf{m}M$.
3: Bestem en tilfældig vektor $\mathbf{e}$ med længde n og vægt w og beregn $\mathbf{y} = \mathbf{m}M + \mathbf{e}$.
4: **return** Cifferskriften $\mathbf{y}$.

I følgende eksempel illustreres brugen af algoritme 10.

3.4.2 Eksempel:

Det vises nu, hvordan Alice omdanner den offentlige nøgle fra eksempel 3.4.1, så det er muligt at sende en meddelelse. Den offentlige nøgle er

$$A_1\pi = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 2 \end{bmatrix} \qquad \text{og} \qquad A_2\pi = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 2 \end{bmatrix}$$

samt $r = 5$ og $t = 1$. Derfor skal $\mathbf{c}_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ dannes, hvormed

$$M_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Alice laver nu et cyklisk skift af denne med $r = 5$ positioner, og dermed opnås at $\mathbf{c}_1' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$, hvormed det ses, at denne ikke genereres af M_1.

Dette betyder, at den skal tilføjes, og derfor bliver

$$M_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Alice ser nu på kodeordet $\mathbf{c}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 \end{bmatrix}$, som er anden række i $A_1\pi, A_2\pi$, og denne tilføjes så

$$M_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 \end{bmatrix}.$$

Herefter skal $\mathbf{c}_2'$ dannes, og bliver $\mathbf{c}_2' = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 \end{bmatrix}$. Det kan dog ses, at $\mathbf{c}_2' = \mathbf{c}_2$, og derfor skal denne ikke tilføjes M_4, og generatormatricen bliver dermed

$$M = M_4 = M_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 \end{bmatrix},$$

og det er nu muligt at indkode en meddelelse ved hjælp af M. Den meddelelse, som Alice ønsker at sende er $\mathbf{m} = \begin{bmatrix} 2 & 0 & 1 \end{bmatrix}$. Nu beregnes

$$\begin{aligned} \mathbf{m}M &= \begin{bmatrix} 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 2 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 2 \end{bmatrix} \end{aligned}$$

og beregningerne er lavet i SAGE og kan ses i appendiks B.12. Fejlvektoren vælges til

$$\mathbf{e} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix},$$

og cifferteksten bliver dermed

$$\begin{aligned} \mathbf{y} &= \mathbf{m}M + \mathbf{e} \\ &= \begin{bmatrix} 0 & 2 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 2 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \end{aligned}$$

som Alice sender til Bob. ◀

Dekodning ved denne form for McEliece kryptosystem gøres ved følgende algoritme.

Nu gives et eksempel på dekodning, når den offentlige nøgle og indkodningen er sket ved Gaborits version af McEliece kryptosystemet.

3.4.3 Eksempel:

I dette eksempel modtager Bob cifferteksten $\mathbf{y} = \begin{bmatrix} 0 & 2 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$ fra eksempel 3.4.2, og han dekoder ved brug af algoritme 11. Punkt 1 i algoritme 11 er at

Algoritme 11 Dekodning i Gaborits version

Input: En vektor $\mathbf{y} = \mathbf{m}M + \mathbf{e} \in \mathbb{F}_q^n$ og den private nøgle (Π, dekodningsalgoritme for C).

Output: Meddelelsen $\mathbf{m}$.

1: Dan generatormatricen M som beskrevet over algoritme 10 ud fra $A_1\pi, A_2\pi, \ldots, A_s\pi$
2: Beregn $\mathbf{y}' = \mathbf{y}\Pi^{-1} = \mathbf{m}M\Pi^{-1} + \mathbf{e}\Pi^{-1}$
3: Benyt dekodningsalgoritmen for C på $\mathbf{y}'$ til at finde $\mathbf{m}$.
4: **return** Meddelelsen $\mathbf{m}$.

danne matricen M ud fra den offentlige nøgle. Udregninger af denne er allerede lavet i eksempel 3.4.2 og kan derfor ses i appendiks B.12.

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Herefter beregnes

$$\mathbf{y}' = \mathbf{y}\Pi^{-1} = \begin{bmatrix} 2 & 2 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Beregningerne kan ses i appendiks B.13. Bob benytter nu en dekodningsalgoritme til at dekode $\mathbf{y}'$ til et kodeord $\mathbf{c}$ med afstand 1 fra cifferteksten. Følgende opnås

$$\mathbf{c} = \begin{bmatrix} 2 & 2 & 2 & 1 & 0 & 0 & 0 & 0 & 2 & 1 \end{bmatrix}.$$

For at opnå meddelelsen $\mathbf{m}$ løser Bob ligningssystemet $\mathbf{m}M\Pi^{-1} = \mathbf{c}$. Udregningerne kan ses i appendiks B.13, og følgende meddelelse opnås.

$$\mathbf{m} = \begin{bmatrix} 2 & 0 & 1 \end{bmatrix}$$

Dette er præcis den meddelelse, Alice sendte i eksempel 3.4.2. ◀

Det er i afsnittet vist, hvordan nøglegenerering, indkodning samt dekodning foregår i Gaborits version. Sikkerheden for dette system betragtes nu. Et af argumenterne, som Gaborit benytter til at sikkerheden i hans version er stor er, at han tager delkoder af quasicykliske koder. Nærmere bestemt er den type koder han benytter delkoder af en kendt BCH-kode, og der eksisterer mange sådanne delkoder. Dette vil nu blive vist, men først skal følgende lemma vises.

3.4.4 Lemma:
Lad C være en r quasicyklisk $[n, k]$-kode, hvor $n = rs$. Så eksisterer en quasicyklisk delkode af orden r med dimension større end eller lig $k - s = k - \frac{n}{r}$, der er fuldstændigt indeholdt i C.

Bevis:
Når C er en r quasicyklisk kode, så er den duale $C^\perp$ også en r quasicyklisk kode. Vælg et vilkårligt kodeord $\mathbf{x}$, der opfylder at $\mathbf{x} \in \mathbb{F}_2^n$ og $\mathbf{x} \notin C^\perp$. Betragt generatormatricen $G_\mathbf{x}$, der er dannet ved at tilføje $\mathbf{x}$ samt de $s - 1$ skift på r positioner af denne til generatormatricen

for $C^{\perp}$. Koden $C_{\mathbf{x}}$, der er genereret af $G_{\mathbf{x}}$, er så en r quasicyklisk kode. Dimensionen af $C_{\mathbf{x}}$ er højest $n-k+s$, da dimensionen af $C^{\perp}$ er $n-k$, og der er tilføjet højst s rækker i konstruktionen af $G_{\mathbf{x}}$. Den duale af $C_{\mathbf{x}}$, koden $C_{\mathbf{x}}^{\perp}$, er en r quasicyklisk kode med dimension mindst $n-(n-k+s)=k-s$, og er derfor fuldstændigt indeholdt i C. ■

Nu vises, at der eksisterer mange delkoder som i lemma 3.4.4. Dette er ifølge Gaborit grundlaget for, at denne version af kryptosystemet er sikker.

3.4.5 Sætning:
Lad C være en r quasicyklisk $[n,k]$-kode. Så kan der ud fra lemma 3.4.4 konstrueres mindst 2^{k-s} forskellige r quasicykliske delkoder.

Bevis:
Antallet af disjunkte quasicykliske delkoder er det samme som antallet af disses duale. Foreningen af alle mulige duale quasicykliske koder må danne hele rummet, og jævnfør lemma 3.4.4 vides, at hver af de duale quasicykliske koder har dimension højest $n-(k-s)=n-k+s$. Dermed kan det mindste antal disjunkte quasicykliske delkoder beregnes ved at tage antallet af elementer i $\mathbb{F}_2^n$ og dele med det maksimale antal af elementer i hver af de duale quasicykliske koder. Det opnås, at der mindst er

$$\frac{2^n}{2^{n-k+s}} = 2^{n-(n-k+s)} = 2^{k-s}$$

forskellige duale quasicykliske koder, og altså at der er mindst 2^{k-s} quasicykliske delkoder. ■

Da 2^{k-s} i praksis er et stort tal, vil det være beregningstekninsk svært at skulle bryde krypteringen, da det er nødvendigt at finde den rigtige kode før selve dekodningen kan påbegyndes, når den private nøgle ikke kendes. Det skal dog nævnes, at det i (Otmani m.fl., 2010) er vist, at Gaborits brug af delkoder af en kendt BCH-kode i visse tilfælde gør det muligt at finde permutationen, og dermed finde strukturen på koden. Idéen er dog stadig interessant, og i dette projekt gives i kapitel 5 en videreudvikling af denne til brug ved matrixprodukt-koder.

Den grundlæggende teori om cykliske og quasicykliske koder er nu gennemgået. Det er i afsnit 3.2 beskrevet, hvordan generatormatricen grundet den quasicykliske struktur kan reduceres i størrelse. Herefter gives en anden repræsentationsform, der muliggør beregning af kodens dimension ved hjælp af Gröbnerbaser. Kapitlet er afrundet med Gaborits version af McEliece kryptosystemet, og netop dette vil være baggrund for den nye version af McEliece kryptosystemet, der præsenteres i dette projekt i kapitel 5.

KAPITEL 4

Matrixprodukt-koder

Dette kapitel bygger på (Hernando m.fl., 2009),(Blackmore m.fl., 2001) samt (Özbudak m.fl., 2002). I dette projekt udvikles en ny version af McEliece kryptosystemet. Med denne version ønskes det, at reducere den offentlige nøglestørrelse. Idéen bag dette er at benytte matrixprodukt-koder samt den tankegang, der ligger til grund for Gaborits version, som præsenteret i foregående afsnit 3.4. I dette kapitel gennemgås den relevante teori omkring matrixprodukt-koder, og den nye version præsenteres i kapitel 5.

Det er i kodningsteori interessant at se på, hvordan der kan dannes koder ud fra andre mindre koder. Det ønskes at se på egenskaberne for disse sammensatte koder ud fra egenskaberne af de mindre. En metode til at sammensætte koder er *produktkoder*. Betragt to koder C_1 og C_2 i $\mathbb{F}_q^n$ med henholdsvis parametrene $[n, k_1, d_1]$ og $[n, k_2, d_2]$. Disse kan sammensættes til en produktkode på følgende måde

$$C' = \{(\mathbf{u}, \mathbf{v}) \mid \mathbf{u} \in C_1, \mathbf{v} \in C_2\}.$$

Dermed bliver C' en $[n', k', d']$-kode, hvor $n' = 2n$ og $k' = k_1 + k_2$. Det ses, at minimumsafstanden afhænger af d_1 og d_2, og denne er givet ved

$$d' = \min\{d(C_1), d(C_2)\}.$$

Dette skyldes, at kodeordene i C' med mindst vægt er dem, hvor enten $\mathbf{u}$ eller $\mathbf{v}$ er nul. Det ønskes, at forbedre minimumsafstanden, da der så kan rettes flere fejl, men for ovenstående konstruktion er dette ikke tilfældet. Derfor arbejdes der ofte med matrixprodukt-koder istedet. Disse er en udvidelse af klassiske opbygninger af nye koder ud fra gamle. Et eksempel på dette er

$$C'' = \{(\mathbf{u}, \mathbf{u} + \mathbf{v}) \mid \mathbf{u} \in C_1, \mathbf{v} \in C_2\},$$

hvilket kaldes *Plotkin-konstruktionen*. Parametrene for denne kode er $[n'', k'', d'']$, hvor længde og dimension er som for koden C'. Minimumsafstanden for denne kode kan potentielt være større. Dette skyldes, at følgende tre muligheder kan forekomme

$$\mathbf{u} = \mathbf{0} \ \wedge \ \mathbf{v} \neq \mathbf{0} \tag{4.1}$$

$$\mathbf{v} = \mathbf{0} \ \wedge \ \mathbf{u} \neq \mathbf{0} \tag{4.2}$$

$$\mathbf{u} \neq \mathbf{0} \ \wedge \ \mathbf{v} \neq \mathbf{0}, \tag{4.3}$$

når der ses bort fra tilfældet $\mathbf{u} = \mathbf{0} \wedge \mathbf{v} = \mathbf{0}$. Ved tilfældet i ligning (4.1) betragtes kodeord på formen $(0, \mathbf{v})$, og vægten af disse må være større end eller lig d_2. I andet tilfælde i ligning (4.2) er kodeordene på formen $(\mathbf{u}, \mathbf{u})$, og disse har altså en vægt, der er større end eller lig $2d_1$. I det sidste tilfælde fra ligning (4.3) arbejdes med kodeord på formen $(\mathbf{u}, \mathbf{u} + \mathbf{v})$. Disse kodeord opfylder, at første del vil have en vægt, der er større end eller lig d_1, men det vides ikke, hvor stor en vægt anden del har. Det vises dog senere, i sætning 4.1.5, at minimumsafstanden for C'' er givet ved

$$d'' \geq \min\{2d(C_1), d(C_2)\},$$

hvilket kan være en forbedring sammenlignet med d'.

En fordel ved matrixprodukt-koder er, at længden af koden ikke er begrænset af q. Dermed kan en større minimumsafstand opnås, hvilket gør det muligt at rette flere fejl. Når q vælges lille er det altså en fordel at arbejde med denne type af koder, men er q stor arbejdes ofte i stedet med koder såsom Reed-Solomon-koder.

4.1 Teori om matrixprodukt-koder

Betragt igen produktkoden C'. Denne kan ses som

$$\begin{bmatrix} \mathbf{u} & \mathbf{v} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{u} & \mathbf{v} \end{bmatrix}.$$

På tilsvarende måde kan Plotkin-konstruktionen betragtes på formen

$$\begin{bmatrix} \mathbf{u} & \mathbf{v} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{u} & \mathbf{u}+\mathbf{v} \end{bmatrix}.$$

Dette kan generaliseres til at indeholde flere koder, hvormed der ikke kun arbejdes med C_1 og C_2, men med s delkoder samt med en matrix, som ikke nødvendigvis er binær og kvadratisk.

4.1.1 Definition:
Lad $A = [a_{ij}]$ være en $s \times \ell$ matrix, hvor $s \leq \ell$, med indgange i $\mathbb{F}_q$ og lad $C_1, \ldots, C_s \subset \mathbb{F}_q^n$ være lineære delkoder med længde n.
*Mængden af alle matrixprodukter $\begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_s \end{bmatrix} \cdot A$, hvor $\mathbf{c}_i \in C_i$ er en $n \times 1$ søjlevektor for $i = 1, 2, \ldots, s$ er **Matrixprodukt-koden** $C = \begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix} \cdot A$.*

Ud fra ovenstående definition er $\mathbf{c}_i = \begin{bmatrix} c_{1,i} & c_{2,i} & \dots & c_{n,i} \end{bmatrix}^T$, og det bemærkes, at et kodeord i matrixprodukt-koden $\mathbf{c} \subseteq \mathbb{F}_q^{ln}$ vil være på formen

$$\begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,s} \\ c_{2,1} & c_{2,2} & \dots & c_{2,s} \\ \vdots & \vdots & \vdots & \vdots \\ c_{n,1} & c_{n,2} & \dots & c_{n,s} \end{bmatrix} \cdot \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,\ell} \\ a_{2,1} & a_{2,2} & \dots & a_{2,\ell} \\ \vdots & \vdots & \vdots & \vdots \\ a_{s,1} & a_{s,2} & \dots & a_{s,\ell} \end{bmatrix} =$$

$$\begin{bmatrix} c_{1,1}a_{1,1} + c_{1,2}a_{2,1} + \ldots + c_{1,s}a_{s,1} & \dots & c_{1,1}a_{1,\ell} + c_{1,2}a_{2,\ell} + \ldots + c_{1,s}a_{s,\ell} \\ c_{2,1}a_{1,1} + c_{2,2}a_{2,1} + \ldots + c_{2,s}a_{s,1} & \dots & c_{2,1}a_{1,\ell} + c_{2,2}a_{2,\ell} + \ldots + c_{2,s}a_{s,\ell} \\ \vdots & \vdots & \vdots \\ c_{n,1}a_{1,1} + c_{n,2}a_{2,1} + \ldots + c_{n,s}a_{s,1} & \dots & c_{n,1}a_{1,\ell} + c_{n,2}a_{2,\ell} + \ldots + c_{n,s}a_{s,\ell} \end{bmatrix}.$$

Det kan ses, at den i'te søjle i kodeordet også kan skrives som følgende sum

$$\sum_{j=1}^{s} a_{j,i}\mathbf{c}_j \in \mathbb{F}_q^n.$$

Nu introduceres *søjlestørrelsesorden*, som er en måde hvor på indgangene i matricer kan arrangeres, så der i stedet bliver tale om en vektor, hvilket er den generelle form for et kodeord. Dette gøres ved, at indgangene i 1. søjle placeres først i vektoren med voksende indeks efter hinanden, herefter indgangene i 2. søjle og så videre. Søjlestørrelsesorden vil nu blive illustreret med et eksempel.

4.1.2 Eksempel:

Følgende $m \times n$ matrix haves.

$$b = \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,n} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{m,1} & b_{m,2} & \dots & b_{m,n} \end{bmatrix}$$

Hvis denne matrix skrives på søjlestørrelsesorden vil dette være en vektor med følgende indgange.

$$\mathbf{b} = \begin{bmatrix} b_{1,1} & b_{2,1} & \dots & b_{m,1} & b_{1,2} & b_{2,2} & \dots & b_{m,2} & \dots & b_{1,n} & b_{2,n} & \dots & b_{m,n} \end{bmatrix}$$

Det ses, at vektor $\mathbf{b}$ har længde mn. ◀

Indgangene i kodeordene, der er $n \times \ell$ matricer, kan ses som vektorer på følgende måde,

$$\mathbf{c} = \left[\sum_{j=1}^{s} a_{j,1}\mathbf{c}_j, \sum_{j=1}^{s} a_{j,2}\mathbf{c}_j, \dots, \sum_{j=1}^{s} a_{j,\ell}\mathbf{c}_j \right] \subseteq \mathbb{F}_q^{n\ell}. \tag{4.4}$$

Nu gives et eksempel på en matrixprodukt-kode, hvor kodeordene illustreres.

4.1.3 Eksempel:

I dette eksempel arbejdes over $\mathbb{F}_2$ med $n = 3$ og $s = 3$. De tre koder C_1, C_2 og C_3 vælges til

$$C_1 = \left\{ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \right\}$$
$$C_2 = \left\{ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \right\}$$
$$C_3 = \left\{ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \right\}.$$

Desuden vælges A som

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Matrixprodukt-koden består altså af $2 \cdot 4 \cdot 2 = 16$ kodeord. Et eksempel på beregning af to af disse er

$$c1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$c2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

De resterende udregninger kan ses i appendiks B.15. Ud fra lignning (4.4) kan kodeordene altså betragtes som følgende vektorer.

$$\mathbf{c}1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{c}2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$
$$\mathbf{c}3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$
$$\mathbf{c}4 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$
$$\mathbf{c}5 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$
$$\mathbf{c}6 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\mathbf{c}7 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$
$$\mathbf{c}8 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$
$$\mathbf{c}9 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$
$$\mathbf{c}10 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$
\begin{aligned}
\mathbf{c}11 &= \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \\
\mathbf{c}12 &= \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \\
\mathbf{c}13 &= \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \\
\mathbf{c}14 &= \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \\
\mathbf{c}15 &= \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \\
\mathbf{c}16 &= \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}
\end{aligned}
$$ ◀

Da $C_1, \ldots, C_s$ er lineære koder kan der findes en generatormatrix G til en matrixprodukt-kode. Denne vil ud fra konstruktionen af koden være,

$$
G = \begin{bmatrix} a_{1,1}G_1 & a_{1,2}G_1 & \ldots & a_{1,\ell}G_1 \\ a_{2,1}G_2 & a_{2,2}G_2 & \ldots & a_{2,\ell}G_2 \\ \vdots & \vdots & \vdots & \vdots \\ a_{s,1}G_s & a_{s,2}G_s & \ldots & a_{s,\ell}G_s \end{bmatrix}, \tag{4.5}
$$

hvor G_i er generatormatrix til koden C_i for $i = 1, \ldots, s$. Der gives nu et eksempel på, hvordan generatormatricen dannes.

4.1.4 Eksempel:

I dette eksempel arbejdes videre med matrixprodukt-koden fra eksempel 4.1.3. Denne matrixprodukt-kode består blandt andet af delkoderne C_1, C_2 og C_3, og de tre tilhørende generatormatricer er henholdsvis

$$
G_1 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \qquad G_2 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad \text{og} \qquad G_3 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}.
$$

Derudover består matrixprodukt-koden af matricen A, der er givet ved

$$
A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.
$$

En generatormatrix for matrixprodukt-koden bliver altså

$$
\begin{aligned}
G &= \begin{bmatrix} a_{1,1}G_1 & a_{1,2}G_1 & a_{1,3}G_1 \\ a_{2,1}G_2 & a_{2,2}G_2 & a_{2,3}G_2 \\ a_{3,1}G_3 & a_{3,2}G_3 & a_{3,3}G_3 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix},
\end{aligned}
$$

og det ses, at alle kodeord fra eksempel 4.1.3 genereres. ◀

Når en delkode C_i har parametrene $[n, k_i, d_i]$, hvor n er længden, k_i er dimensionen og d_i er minimumsafstanden, og matricen A har fuld rang, vil parametrene for C være som i sætning 4.1.5.

Inden denne sætning er det dog nødvendigt at indføre en notation for rækkerne i A. Den i'te række i A, hvor $i = 1, 2, \ldots, s$ noteres $R_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,\ell}) \in \mathbb{F}_q^\ell$. Koden C_{R_i} er genereret af $\langle R_1, R_2, \ldots, R_i \rangle$, og minimumsafstanden for denne kode noteres D_i.

4.1.5 Sætning:
En matrixprodukt-kode $C = \begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix} \cdot A\}$, har følgende parametre.

1. *Kodens længde er givet ved, længde(C) = længde(C_{R_i}) · længde$(C_i) = \ell n$*
2. *Dimensionen for koden er $k = dim(C) = \sum_{i=1}^{s} k_i$, hvor k_i er som beskrevet ovenfor*
3. *Minimumsafstanden opfylder $d(C) \geq \min\{d_1 D_1, d_2 D_2, \ldots, d_s D_s\}$, hvor $d_i = d(C_i)$ og $D_i = d(C_{R_i})$, hvor C_{R_i} er som beskrevet ovenfor.*

Bevis:
Først vises punkt 1. Det ses, at $\mathbf{c} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_s \end{bmatrix} \cdot A \in C$. Det vides, at $\begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_s \end{bmatrix}$ er en $n \times s$ matrix og A en $s \times \ell$ matrix. Dermed er $\mathbf{c}$ en $n \times \ell$ matrix, og kan ved brug af søjlestørrelsesorden ses som en vektor i $\mathbb{F}_q^{\ell n}$. Dermed er C en lineær kode med længde ℓn.

Nu vises punkt 2. Den i'te række i $\begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_s \end{bmatrix}$ noteres $\mathbf{c}^{(i)} \in \mathbb{F}_q^s$, hvormed

$$\begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_s \end{bmatrix} \cdot A = \begin{bmatrix} \mathbf{c}^{(1)} \cdot A \\ \mathbf{c}^{(2)} \cdot A \\ \vdots \\ \mathbf{c}^{(n)} \cdot A \end{bmatrix}$$

Det haves, at $\mathbf{c}^{(i)} \cdot A \in C$ når $1 \leq i \leq n$. Da A har fuld rang, og $\ell \geq s$, er rækkerne heri lineært uafhængige, hvilket medfører, at hvis $\begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_s \end{bmatrix} \neq 0$, så er $\begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_s \end{bmatrix} \cdot A \neq 0$. Da koden $C \subseteq \mathbb{F}_q^{\ell n}$ haves dimensionen af denne

$$k = dim(C) = dim(\begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix}).$$

Da $\begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix}$ er et lineært vektorrum haves det yderligere, at

$$dim(\begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix}) = dim(C_1) + dim(C_2) + \ldots + dim(C_s) = \sum_{i=1}^{s} k_i.$$

For at vise punkt 3 ses på et kodeord i C, der har formen $\mathbf{c} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_s \end{bmatrix} \cdot A$. Den j'te række i $\begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_s \end{bmatrix}$ skrives som $\begin{bmatrix} c_{j,1} & c_{j,2} & \ldots & c_{j,s} \end{bmatrix}$ for alle $j = 1, 2, \ldots, n$, hvor

$c_{j,i}$ er den j'te indgang i kodeordet $\mathbf{c}_i$. Lad $\mathbf{c}_r \neq 0$ og $\mathbf{c}_i = 0 \; \forall \; i > r$, så er

$$\begin{aligned}
\mathbf{c} &= \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_s \end{bmatrix} \cdot A \\
&= \begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,r} & 0 & \dots & 0 \\ c_{2,1} & c_{2,2} & \dots & c_{2,r} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{n,1} & c_{n,2} & \dots & c_{n,r} & 0 & \dots & 0 \end{bmatrix} \cdot \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,\ell} \\ a_{2,1} & a_{2,2} & \dots & a_{2,\ell} \\ \vdots & \vdots & \vdots & \vdots \\ a_{s,1} & a_{s,2} & \dots & a_{s,\ell} \end{bmatrix} \\
&= \begin{bmatrix} c_{1,1}a_{1,1} + c_{1,2}a_{2,1} + \ldots + c_{1,r}a_{r,1} & \dots & c_{1,1}a_{1,\ell} + c_{1,2}a_{2,\ell} + \ldots + c_{1,r}a_{r,\ell} \\ c_{2,1}a_{1,1} + c_{2,2}a_{2,1} + \ldots + c_{2,r}a_{r,1} & \dots & c_{2,1}a_{1,\ell} + c_{2,2}a_{2,\ell} + \ldots + c_{2,r}a_{r,\ell} \\ \vdots & \vdots & \vdots \\ c_{n,1}a_{1,1} + c_{n,2}a_{2,1} + \ldots + c_{n,r}a_{r,1} & \dots & c_{n,1}a_{1,\ell} + c_{n,2}a_{2,\ell} + \ldots + c_{n,r}a_{r,\ell} \end{bmatrix},
\end{aligned}$$

hvormed $\mathbf{c}$ kun indeholder komponenter fra A op til række R_r. Dermed må $\begin{bmatrix} c_{j,1} & c_{j,2} & \dots & c_{j,s} \end{bmatrix} \cdot A \in C_{R_r} \forall j$. Da $\mathbf{c}_r \neq 0$ må der være mindst $d(C_r) = d_r$ indgange forskellige fra nul heri.

Lad $c_{i_v,r} \neq 0$ for $v = 1, 2, \ldots, d_r$, så haves, at produktet $\begin{bmatrix} c_{i_v,1} & c_{i_v,2} & \dots & c_{i_v,s} \end{bmatrix} \cdot A$, der er indeholdt i C_{R_r} er forskelligt fra 0, da A har fuld rang. Vægten af denne må derfor være større end eller lig D_r, og da C_r har minimumsafstand d_r, må $d(\mathbf{c}) \geq d_r D_r$. Derfor haves det, at

$$d(C) \geq \min\{d_1 D_1, d_2 D_2, \ldots, d_s D_s\}. \qquad \blacksquare$$

Parametrene for en matrixprodukt-kode er nu vist. Da C_{R_i} dannes ud fra rækkerne $R_1, R_2, \ldots, R_i$ må minimumsafstanden, D_i falde jo større i bliver. Dermed haves at $D_1 \geq D_2 \geq \ldots \geq D_s$, hvorfor det vil være en fordel at vælge en kode C_j med en minimumsafstand, der er større end eller lig den for C_{j-1}. Ud fra dette er det også en fordel at vælge C_j med dimension mindre end eller lig den for C_{j-1}. Det illustreres nu ved hjælp af et eksempel, hvordan parametrene for en matrixprodukt-kode kan være.

4.1.6 Eksempel:

De lineære delkoder $C_1, C_2, C_3 \in \mathbb{F}_3$ har henholdsvis parametrene $[4, 4, 1]$, $[4, 3, 2]$ og $[4, 2, 3]$. Det ønskes at finde parametre for matrixprodukt-koden $C = \begin{bmatrix} C_1 & C_2 & C_3 \end{bmatrix} \cdot A$, hvor

$$A = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Det ses, at A har fuld rang, hvorfor sætning 4.1.5 kan benyttes. Først beregnes længden af matrixprodukt-koden C.

$$\text{længde}(C) = \text{længde}(C_{R_i}) \cdot \text{længde}(C_i) = 4 \cdot 4 = 16$$

Dimensionen af C er $k = \sum_{i=1}^{s} k_i$, hvorfor

$$k = \sum_{i=1}^{3} k_i = 4 + 3 + 2 = 9.$$

Minimumsafstanden kan ikke findes eksakt, men der haves derimod en nedre grænse for, hvad den kan være. Det ses fra matrix A, at $D_1 = 4$, $D_2 = 2$ og $D_3 = 1$, hvormed

$$d(C) \geq \min\{d_1D_1, d_2D_2, d_3D_3\} = \min\{1 \cdot 4, 2 \cdot 2, 3 \cdot 1\} = \min\{4, 4, 3\}.$$

Derfor må $d(C) \geq 3$. ◀

Det er muligt at opfylde betingelserne $d(C_j) \geq d(C_{j-1})$ og $k_j \leq k_{j-1}$ ved at $C_1, C_2, \ldots, C_s$ er *indlejrede koder*, hvilket betyder, at $C_1 \supset C_2 \supset \ldots \supset C_s$. Yderligere betyder dette, at minimumsafstanden af C_i vil være mindre end eller lig den for C_{i+1}, altså $d_1 \leq d_2 \leq \ldots \leq d_s$. Med denne type koder, vil minimumsafstanden være $d(C) = \min\{d_1D_1, d_2D_2, \ldots, d_sD_s\}$, hvilket vises i sætning 4.1.7. I teorien er det altid en fordel, at minimumsafstanden kan være større, så der eventuelt kan rettes flere fejl. I praksis er det en fordel at vide, præcis hvor stor minimumsafstanden er, for så vides, hvor mange fejl der kan rettes.

4.1.7 Sætning:
Lad C være matrixprodukt-koden $\begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix} \cdot A$, hvor $C_1, C_2, \ldots, C_s$ er indlejrede koder og A er en $s \times \ell$ matrix i $\mathbb{F}_q$ med fuld rang. Så er minimumsafstanden af koden C givet ved

$$d(C) = \min\{d_1D_1, d_2D_2, \ldots, d_sD_s\},$$

hvor $d_i = d(C_i)$ og $D_i = d(C_{R_i})$, med C_{R_i} som beskrevet tidligere.

Bevis:
Det er i sætning 4.1.5 vist at minimumsafstanden for C opfylder uligheden

$$d(C) \geq \min\{d_1D_1, d_2D_2, \ldots, d_sD_s\}.$$

For at vise, at der gælder lighedstegn dannes et kodeord med vægt d_rD_r for $r = 1, 2, \ldots, s$. Da $C_1, C_2, \ldots, C_s$ er indlejrede, er det muligt at vælge $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_r$, på en sådan måde at $\mathbf{c}_1 = \mathbf{c}_2 = \ldots = \mathbf{c}_r$ med vægt d_r. Yderligere vælges $\mathbf{c}_{r+1}, \ldots, \mathbf{c}_s$ til at være $\mathbf{0}$.

Lad $f = \sum_{j=1}^{r} f_iR_i$ med $f_i \in \mathbb{F}_q$ være et ord i C_{R_i} med vægt D_r. Hvis $\mathbf{c}'_i = \mathbf{c}_if_i$ haves, så kan et kodeord i C ud fra ligning (4.4) skrives som

$$\left[\sum_{j=1}^{s} a_{j,1}\mathbf{c}'_j, \sum_{j=1}^{s} a_{j,2}\mathbf{c}'_j, \ldots, \sum_{j=1}^{s} a_{j,\ell}\mathbf{c}'_j\right] = \left[\sum_{j=1}^{s} a_{j,1}\mathbf{c}_jf_j, \sum_{j=1}^{s} a_{j,2}\mathbf{c}_jf_j, \ldots, \sum_{j=1}^{s} a_{j,\ell}\mathbf{c}_jf_j\right].$$

Da $\mathbf{c}_j = 0$ for $j > r$ er leddene i summerne for $j > r$ lig nul, og dermed kan summerne ændres, så der i stedet summeres fra $j = 1, \ldots, r$. Derudover er $\mathbf{c}_1 = \ldots = \mathbf{c}_r$, hvormed $\mathbf{c}_j$ kan ændres til $\mathbf{c}_1$, og følgende haves

$$\begin{bmatrix} \sum_{j=1}^{r} a_{j,1}\mathbf{c}_1f_j & \sum_{j=1}^{r} a_{j,2}\mathbf{c}_1f_j & \ldots & \sum_{j=1}^{r} a_{j,\ell}\mathbf{c}_1f_j \end{bmatrix}$$
$$= \mathbf{c}_1 \begin{bmatrix} \sum_{j=1}^{r} a_{j,1}f_j & \sum_{j=1}^{r} a_{j,2}f_j & \ldots & \sum_{j=1}^{r} a_{j,\ell}f_j \end{bmatrix}.$$

Betragtes indgangene $a_{j,1}, a_{j,2}, \ldots, a_{j,\ell}$ ses, at $\begin{bmatrix} a_{j,1} & a_{j,2} & \ldots & a_{j,\ell} \end{bmatrix}$ er en rækkevektor i A, og den kan derfor skrives som R_j, hvorfor

$$\mathbf{c}_1 \left[\sum_{j=1}^{r} a_{j,1} f_j, \sum_{j=1}^{r} a_{j,2} f_j, \ldots, \sum_{j=1}^{r} a_{j,\ell} f_j \right] = \mathbf{c}_1 \left[\sum_{j=1}^{r} R_j f_j \right] = \mathbf{c}_1 f.$$

Dette er et kodeord i C med vægt $d_r D_r$, dermed haves

$$d(C) = \min\{d_1 D_1, d_2 D_2, \ldots, d_s D_s\}. \qquad \blacksquare$$

Nu gives et eksempel på brugen af sætning 4.1.7, hvor minimumsafstanden for en matrixprodukt-kode findes.

4.1.8 Eksempel:
I dette eksempel arbejdes over $\mathbb{F}_3$. Der arbejdes med matricen

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

samt med fire indlejrede koder $C_1 \supset C_2 \supset C_3 \supset C_4$, der er genereret ud fra følgende generatormatricer

$$G_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}, \qquad G_2 = G_3 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 \end{bmatrix} \qquad \text{og} \qquad G_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}.$$

Det ses, at C_1 er en $[4,3]$-kode, C_2 og C_3 er $[4,2]$-koder, mens C_4 er en $[4,1]$-kode. Kodernes minimumsafstande er henholdsvis d_1, d_2, d_3 og d_4, og disse er beregnet til $d_1 = 2$, $d_2 = 2$, $d_3 = 2$ og $d_4 = 4$. Udregningerne er foretaget i SAGE og kan ses i appendiks B.16. Ud over minimumsafstandene for C_1, C_2, C_3 og C_4 skal minimumsafstandene D_1, D_2, D_3 og D_4 for koderne C_{R_1}, C_{R_2}, C_{R_3} og C_{R_4} også benyttes, når matrixprodukt-kodens minimumsafstand skal beregnes. Koderne er som følger

$$C_{R_1} = \left\langle \begin{bmatrix} 1 & 2 & 1 & 0 \end{bmatrix} \right\rangle$$

$$C_{R_2} = \left\langle \begin{bmatrix} 1 & 2 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 2 & 0 & 0 \end{bmatrix} \right\rangle$$

$$C_{R_3} = \left\langle \begin{bmatrix} 1 & 2 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 2 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 2 & 2 \end{bmatrix} \right\rangle$$

$$C_{R_4} = \left\langle \begin{bmatrix} 1 & 2 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 2 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 2 & 2 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \right\rangle,$$

og deres minimumsafstande er $D_1 = 3$, $D_2 = 1$, $D_3 = 1$ og $D_4 = 1$. Disse er beregnet i SAGE og udregningerne kan ses i appendiks B.16. Matrixproduk-kodens minimumsafstand

bliver altså, jævnfør sætning 4.1.7,

$$\begin{aligned} d(C) &= \min\{2 \cdot 3, 2 \cdot 1, 2 \cdot 1, 4 \cdot 1\} \\ &= \min\{6, 2, 2, 4\}. \end{aligned}$$

Dermed bliver minimumsafstanden for matrixprodukt-koden $d(C) = 2$. ◀

Der er nu gennemgået udvalgte egenskaber for matrixprodukt-koder. En vigtig del af McEliece kryptosystemet er, at der eksisterer en effektiv dekodningsalgoritme til den kodetype, der benyttes. Dette kapitel præsenterer en dekodningalgoritme for matrixprodukt-koder, hvor det kræves, at matricen A opfylder følgende definition.

4.1.9 Definition:
Lad A være en $s \times \ell$ matrix, og A_t være matricen bestående af de første t rækker af A. Med $A(j_1, j_2, \ldots, j_t)$ hvor $1 \leq j_1 \leq j_2 \leq \ldots \leq j_t \leq \ell$ noteres $t \times t$ matricen bestående af søjlerne $j_1, j_2, \ldots j_t$ i A_t. En matrix A siges at være ikke-singulær ved søjler hvis $A(j_1, j_2, \ldots, j_t)$ er ikke-singulær for hvert $1 \leq t \leq s$ og $1 \leq j_1 \leq j_2 \leq \ldots \leq j_t \leq \ell$. En ikke-singlær ved søjler matrix A har fuld rang.

4.1.10 Sætning:
A er ikke-singulær ved søjler hvis og kun hvis C_{R_i} er MDS for alle i.

Bevis:
Når A er ikke-singulær ved søjler haves det fra definition 4.1.9, at $A(j_1, j_2, \ldots, j_t)$ er ikke-singulær for hvert $1 \leq t \leq s$ og $1 \leq j_1 \leq j_2 \leq \ldots \leq j_t \leq \ell$. Dette betyder, at determinanten af $A(j_1, j_2, \ldots, j_t)$ opfylder, at $\det(A(j_1, j_2, \ldots, j_t)) \neq 0$ for alle t.

Generatormatricen for C_{R_i}, der er koden genereret af de første i rækker af A, kan vælges til at være A_i. Dette betyder, at matricen A_i har fuld rang, da $s \leq \ell$ og det vides fra lineær algebra, at $\det(A_i) \neq 0$ hvis og kun hvis A_i har fuld rang.

Når matricen har fuld rang, så er søjlerne lineært uafhængige. Korollar 3 i kapitel 11 fra (MacWilliams m.fl., 1977) giver, at søjlerne i generatormatricen er lineært uafhængige hvis og kun hvis C_{R_i} er MDS. ■

Når matricen A er ikke-singulær ved søjler gælder sætning 4.1.13 for minimumsafstanden af matrixprodukt-koden. For at vise denne sætning, er det nødvendigt at have følgende to propositioner.

4.1.11 Proposition:
Lad A være en ikke-singulær ved søjler matrix. Så har A højst $i - 1$ nul-indgange i den i'te række for $1 \leq i \leq s$. Hvis A yderligere er triangulær gælder, at den har nøjagtig $i - 1$ nul-indgange i den i'te række for $1 \leq i \leq s$.

Bevis:
Antag for modstrid, at der i den i'te række i A er mere end $i - 1$ nul-indgange. Hvis $a_{1,j_1}, \ldots, a_{i,j_i}$ er nul-indgange, så er $A(j_1, \ldots, j_i)$ singulær, da $\det(A(j_1, \ldots, j_i)) = 0$.

Dermed er A ikke ikke-singulær ved søjler jævnfør definition 4.1.9. Når A er triangulær vides, at der er mindst $i-1$ nul-indgange i den i'te række. Dermed må der i en triangulær ikke-singulær ved søjler matrix være nøjagtig $i-1$ nul-indgange i i'te række. ■

4.1.12 Proposition:

1. *Hvis A_ρ er en rækkepermutation af A, så er*

$$\begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix} \cdot A = \begin{bmatrix} C_{\rho(1)} & C_{\rho(2)} & \cdots & C_{\rho(s)} \end{bmatrix} \cdot A_\rho.$$

2. *Hvis A_κ er en søjlepermutation af A, så er*

$$\begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix} \cdot A_\kappa \equiv \begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix} \cdot A.$$

Bevis:
Punkt 1 i proposition 4.1.12 følger, da delkodernes kodeord $\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_s$ svarer til søjler, og disse permuteres tilsvarende med rækkerne i A. Dermed bliver det de samme indgange, der multipliceres. Yderligere vil det også være de samme indgange i kodeordet, der opnås, da summerne er ens, men leddenes rækkefølge er ændret.

Punkt 2 følger, da en søjlepermutation af A blot medfører, at søjlerne i den samlede matrixprodukt-kode permuteres. Dermed opnås altså en ækvivalent kode, det vil sige en kode med samme parametre $[n, k, d]$. ■

Nu kan følgende sætning vises.

4.1.13 Sætning:
Lad $C = \begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix} \cdot A$ og lad A være ikke-singulær ved søjler. Så gælder, at

$$d(C) \geq d^* = \min\{\ell d_1, (\ell-1)d_2, \ldots, (\ell-s+1)d_s\} \tag{4.6}$$

Yderligere gælder, at hvis A er triangulær, så

$$d(C) = d^* = \min\{\ell d_1, (\ell-1)d_2, \ldots, (\ell-s+1)d_s\} \tag{4.7}$$

Bevis:
Fra sætning 4.1.5 punkt 3 vides at $d(C) \geq \min\{d_1 D_1, d_2 D_2, \ldots, d_s D_s\}$. Når A er ikke-singulær ved søjler siger sætning 4.1.10 at C_{R_i} er MDS, og dermed at

$$D_i = n - k + 1 = \ell - i + 1$$

jævnfør Singletongrænsen. Heraf følger ulighed (4.6).

Nu vises ligning (4.7). Det kan uden tab af generalitet antages, at A er øvretriangulær, da søjlepermutation af A danner en ækvivalent kode jævnfør proposition 4.1.12. Det skal vises, at der findes to kodeord i C med afstand højst d^* til hinanden, det vil sige $d(C) \leq d^*$.

Lad $(\ell-k+1)d_k$ have den mindste værdi af $\ell d_1, (\ell-1)d_2, \ldots, (\ell-s+1)d_s$, og altså svare til d^*. Lad $\mathbf{c}$ og $\mathbf{c}'$ være to kodeord i $C = \begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix}$, hvor $\mathbf{c}_k, \mathbf{c}'_k \in C_k$ og $\mathbf{c}_i = \mathbf{c}'_i \in C_i$ for $i \neq k$. Derudover skal afstanden mellem $\mathbf{c}$ og $\mathbf{c}'$ være d_k, $d(\mathbf{c}, \mathbf{c}') = d_k$. Det vides, at sådan to kodeord eksisterer, da minimumsafstanden i C_k er givet ved d_k. Da de to kodeord $\mathbf{c}$ og $\mathbf{c}'$ er ens bortset fra den del, der består af henholdsvis $\mathbf{c}_k$ og $\mathbf{c}'_k$, er det kun den k'te række i A, der har relevans, da afstanden de andre steder er nul. Det vides, at A er en øvretriangulær matrix, og dermed gælder, at de $k-1$ første indgange i k'te række i A er nul. Altså opnås følgende.

$$\begin{aligned} d(\mathbf{c}A, \mathbf{c}'A) &= d(\mathbf{c}_k a_{k,k}, \mathbf{c}'_k a_{k,k}) + d(\mathbf{c}_k a_{k,k+1}, \mathbf{c}'_k a_{k,k+1}) + \ldots + d(\mathbf{c}_k a_{k,\ell}, \mathbf{c}'_k a_{k,\ell}) \\ &\leq (\ell - (k-1))d_k \\ &= (\ell - k + 1)d_k \\ &= d^* \end{aligned}$$

Fra ligning (4.6) vides, at $d(C) \geq d^*$, og det er nu vist, at $d(C) \leq d^*$. Dermed må der gælde lighed, og ligning (4.7) er vist. ■

Bemærk, at hvis delkoderne $C_1, C_2, \ldots, C_s$ er indlejrede på følgende måde

$$C_1 \supset C_2 \supset \ldots \supset C_s$$

og A er ikke-singulær ved søjler, gælder

$$d(C) = d^* = \min\{\ell d_1, (\ell-1)d_2, \ldots, (\ell-s+1)d_s\}. \tag{4.8}$$

Dette følger som en direkte konsekvens af sætning 4.1.10 og sætning 4.1.7.

I dette afsnit er den grundlæggende teori om matrixprodukt-koder gennemgået, og det er netop denne teori, der danner grundlag for følgende afsnit.

4.2 McEliece kryptosystemet med matrixprodukt-koder

I dette afsnit gives en algoritme til dekodning af matrixprodukt-koder, hvilket er et krav for at kunne benytte denne type koder til McEliece kryptosystemet. Igennem afsnittet vil være et gennemgående eksempel for McEliece kryptosystemet med matrixprodukt-koder, hvor både nøglegenerering, indkodning samt dekodning gennemgås. I den algoritme, algoritme 12, der præsenteres til dekodning af matrixprodukt-koder er der en række krav, som skal være opfyldt. Det skal gælde, at de s delkoder $C_1, C_2, \ldots, C_s \subset \mathbb{F}_q^n$ er indlejrede, samt at matricen A er ikke-singulær ved søjler. Der gives nu et eksempel på McEliece nøglegenerering af en matrixprodukt-kode, hvor delkoderne er Reed-Solomon-koder og ovenstående krav er opfyldt.

4.2.1 Eksempel:

Bob benytter algoritme 1 til at generere den offentlige og private nøgle. Der haves tre Reed-Solomon-koder over $\mathbb{F}_{17}$ med parametrene $C_1 = RS_{17}[16, 10]$, $C_2 = RS_{17}[16, 9]$ og $C_3 = RS_{17}[16, 8]$. Det er dermed muligt for koderne at rette henholdsvis $t_1 = 3$, $t_2 = 3$ og

$t_3 = 4$ fejl. Først skal der findes et primitivt element i $\mathbb{F}_{17}$, og herefter skal punkterne x_i, som Reed-Solomon-koden skal evalueres i, genereres. Disse genereres som $x_i = \beta^{i-1}$ for $i = 1, 2, \ldots, n$, hvor β er det primitive element. Dette er gjort i SAGE, og beregningerne af dette kan ses i appendiks B.17. Herefter skal Bob danne generatormatricerne for C_1, C_2 og C_3. Disse bliver

$$G_1 = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 3 & 9 & 10 & 13 & 5 & 15 & 11 & 16 & 14 & 8 & 7 & 4 & 12 & 2 & 6 \\
1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 & 1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 \\
1 & 10 & 15 & 14 & 4 & 6 & 9 & 5 & 16 & 7 & 2 & 3 & 13 & 11 & 8 & 12 \\
1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 \\
1 & 5 & 8 & 6 & 13 & 14 & 2 & 10 & 16 & 12 & 9 & 11 & 4 & 3 & 15 & 7 \\
1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 & 1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 \\
1 & 11 & 2 & 5 & 4 & 10 & 8 & 3 & 16 & 6 & 15 & 12 & 13 & 7 & 9 & 14 \\
1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 \\
1 & 14 & 9 & 7 & 13 & 12 & 15 & 6 & 16 & 3 & 8 & 10 & 4 & 5 & 2 & 11
\end{bmatrix}$$

$$G_2 = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 3 & 9 & 10 & 13 & 5 & 15 & 11 & 16 & 14 & 8 & 7 & 4 & 12 & 2 & 6 \\
1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 & 1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 \\
1 & 10 & 15 & 14 & 4 & 6 & 9 & 5 & 16 & 7 & 2 & 3 & 13 & 11 & 8 & 12 \\
1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 \\
1 & 5 & 8 & 6 & 13 & 14 & 2 & 10 & 16 & 12 & 9 & 11 & 4 & 3 & 15 & 7 \\
1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 & 1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 \\
1 & 11 & 2 & 5 & 4 & 10 & 8 & 3 & 16 & 6 & 15 & 12 & 13 & 7 & 9 & 14 \\
1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16
\end{bmatrix}$$

$$G_3 = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 3 & 9 & 10 & 13 & 5 & 15 & 11 & 16 & 14 & 8 & 7 & 4 & 12 & 2 & 6 \\
1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 & 1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 \\
1 & 10 & 15 & 14 & 4 & 6 & 9 & 5 & 16 & 7 & 2 & 3 & 13 & 11 & 8 & 12 \\
1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 \\
1 & 5 & 8 & 6 & 13 & 14 & 2 & 10 & 16 & 12 & 9 & 11 & 4 & 3 & 15 & 7 \\
1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 & 1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 \\
1 & 11 & 2 & 5 & 4 & 10 & 8 & 3 & 16 & 6 & 15 & 12 & 13 & 7 & 9 & 14
\end{bmatrix}.$$

Da der er tre koder i matrixprodukt-koden skal matricen A være $3 \times \ell$. Det ønskes senere at dekode matrixprodukt-koden, og til dette benyttes algoritme 12. Denne har som et krav, at matricen A skal være ikke-singulær ved søjler. Dette kan sikres ved at vælge $\ell = 3$, hvormed betingelsen $s \leq \ell$ er opfyldt og A vælges til

$$A = \begin{bmatrix} 7 & 5 & 11 \\ 0 & 10 & 9 \\ 0 & 0 & 3 \end{bmatrix}.$$

Herefter kan Bob danne generatormatricen for matrixprodukt-koden ud fra ligning (4.5). Generatormatricen for koden bliver en 27×48 matrix. Denne samt beregningerne af G_1, G_2 og G_3 kan ses i appendiks B.17. Punkt 1 og 2 i algoritme 1 er nu gjort, og der skal herefter i punkt 3 og 4 dannes henholdsvis en tilfældig $n \times n = 48 \times 48$ permutationsmatrix P samt en tilfældig $k \times k = 27 \times 27$ invertibel matrix S. Disse matricer er også beregnet i SAGE, og kan ses i appendiks B.17. I punkt 5 skal $\hat{G} = SGP$ beregnes. Dette bliver en 27×48 matrix og på grund af størrelsen vil denne ikke blive printet her. Det er dog muligt at se beregningerne af $\hat{G}$ i appendiks B.17. Minimumsafstanden for matrixprodukt-koden skal også beregnes, og da der arbejdes med indlejrede koder haves fra sætning 4.1.7, at denne er givet ved

$$\begin{aligned} d(C) &= \min\{d_1 D_1, d_2 D_2, \ldots, d_s D_s\} \\ &= \min\{7 * 3, 8 * 2, 9 * 1\} \\ &= 9. \end{aligned}$$

Dermed er det muligt at rette $t = 4$ fejl. Bob har nu genereret den offentlige nøgle $\hat{G}$ og t samt den private nøgle bestående af G, P, S og en dekodningsalgoritme til C. ◄

Når den offentlige nøgle er genereret skal Alice nu benytte denne til at indkode en meddelelse til Bob. Dette gøres i følgende eksempel.

4.2.2 Eksempel:
Den offentlige nøgle fra eksempel 4.2.1 benyttes. Meddelelsen som Alice ønsker at sende skal have længde 27, og denne er

$$m = [\ 11\ 15\ 4\ 14\ 9\ 14\ 7\ 0\ 13\ 5\ 4\ 0\ 3\ 1\ 13\ 9\ 12\ 12\ 1\ 0\ 15\ 7\ 0\ 9\ 4\ 1\ 0\ ]. \tag{4.9}$$

Alice benytter algoritme 2 til at indkode meddelelsen. Det første hun skal gøre i punkt 1 er at beregn $m\hat{G}$. Dette gøres i SAGE, hvor $\hat{G}$ fra eksempel 4.2.1 benyttes, og beregningerne kan ses i appendiks B.18.

$$\begin{aligned} \mathbf{m}\hat{G} = [\ &8\ 16\ 14\ 4\ 11\ 14\ 0\ 14\ 16\ 14\ 5\ 11\ 6\ 6\ 1\ 1\ 11\ 16\ 7\ 4\ 14\ 9\ 13\ 1\ 11\ 16\ 1\ 5\ 13\ 3 \\ &5\ 9\ 9\ 9\ 14\ 12\ 11\ 2\ 3\ 2\ 2\ 14\ 13\ 15\ 10\ 4\ 16\ 3\ ] \end{aligned}$$

Herefter skal Alice vælge en fejlvektor med vægt højest $t = 4$.

$$\begin{aligned} e = [\ &0\ 0\ 0\ 0\ 0\ 0\ 0\ 3\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 7\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 15\ 0\ 2 \\ &0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ ] \end{aligned}$$

Når dette er gjort skal $\mathbf{y}$ dannes i algoritmens punkt 2.

$$\begin{aligned} \mathbf{y} &= \mathbf{m}\hat{G} + \mathbf{e} \\ &= [\ 8\ 16\ 14\ 4\ 11\ 14\ 0\ \underline{0}\ 16\ 14\ 5\ 11\ 6\ 6\ 1\ 1\ 11\ 16\ \underline{14}\ 4\ 14\ 9\ 13\ 1\ 11\ 16\ 1\ 5\ 13\ 3\ 5\ 9\ \underline{7}\ 9 \\ &\quad \underline{16}\ 12\ 11\ 2\ 3\ 2\ 2\ 14\ 13\ 15\ 10\ 4\ 16\ 3\ ] \end{aligned}$$

Alice har nu benyttet algoritme 2 til at indkode meddelelsen, og hun kan nu sende cifferteksten $\mathbf{y}$ til Bob. ◄

Når det ønskes at dekode er der som tidligere nævnt et krav om, at der kendes en effektiv dekodningsalgoritme for den valgte kodetype. Da der i dette kapitel arbejdes med matrixprodukt-koder, er det altså et krav, at der eksisterer en effektiv dekodningsalgoritme for disse koder.

Da der arbejdes med indlejrede koder $C_1 \supset C_2 \supset \ldots \supset C_s$ må hver del $\sum_{j=1}^{s} a_{j,i}\mathbf{c}_j$ af kodeordet $\mathbf{c}$ være indeholdt i koden C_1. Dermed er en intuitiv metode til dekodning at benytte dekodningsalgoritmen DC_1 til at dekode hver del, men med denne metode kan det ikke garanteres, at der kan rettes det optimale antal fejl. Koden C kan potentielt rette $t < \frac{d}{2}$ fejl, og problematikken opstår, da minimumsafstanden i C_1 er mindre end eller lig minimumsafstanden for C_2, C_3 og så videre. Derfor benyttes en anden metode, der udnytter kodens fejlkorrigerende potentialer. Ved denne metode benyttes dekodningsalgoritmerne DC_i for $i = 1, \ldots, s$, og der kan rettes op til $\frac{d(C)-1}{2}$ fejl, hvor $d(C)$ er som i ligning (4.8). Nu præsenteres en sådan algoritme.

Algoritme 12 Dekodning af indlejrede matrixprodukt-koder

Input: En ciffertekst $\mathbf{y} \in \mathbb{F}_q^{n\ell}$ hvor $\mathbf{y} = \mathbf{c} + \mathbf{e}$ og $w(\mathbf{e}) \leq t$, $C_1 \supset C_2 \supset \ldots \supset C_s$, og A en ikke-singulær ved søjler matrix. Yderligere haves DC_i, en effektiv dekodningsalgoritme for C_i for $i = 1, \ldots, s$

Output: Et kodeord $\mathbf{c} \in \mathbb{F}_q^{n\ell}$.

1: $\mathbf{y}' = \mathbf{y}; \quad A' = A$
2: **for** $\{i_1, \ldots, i_s\} \subset \{1, \ldots, \ell\}$ **do**
3: $\quad \mathbf{y} = \mathbf{y}'; \quad A = A'$
4: $\quad$ **for** $j = 1, \ldots, s$ **do**
5: $\quad\quad y_{i_j} = DC_j(y_{i_j})$
6: $\quad\quad$ **if** $y_{i_j} =$ *„failure“* **then**
7: $\quad\quad\quad$ **STOP** og vælg et andet $i_1, \ldots, i_s$ i punkt 2
8: $\quad\quad$ **end if**
9: $\quad\quad$ **for** $k = j+1, \ldots, s$ **do**
10: $\quad\quad\quad \mathbf{y}_{ik} = \mathbf{y}_{ik} - \frac{a_{j,i_k}}{a_{j,i_j}}\mathbf{y}_{ij}$
11: $\quad\quad\quad col_{i_k}(A) = col_{i_k}(A) - \frac{a_{j,i_k}}{a_{j,i_j}} col_{i_j}(A)$
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: $\quad$ Dan $\begin{bmatrix} c_1 & \ldots & c_s \end{bmatrix}$ ud fra $y_{i_1}, \ldots, y_{i_s}$
15: $\quad \mathbf{y} = \begin{bmatrix} C_1 & \ldots & C_s \end{bmatrix} \cdot A$ ud fra ligning (4.4)
16: $\quad$ **if** $\mathbf{y} \in C$ og $w(\mathbf{y} - \mathbf{y}') \leq t$ **then**
17: $\quad\quad$ **return** Kodeordet $\mathbf{y} = \mathbf{c}$.
18: $\quad$ **end if**
19: **end for**

I algoritme 12 antages, at der kendes en effektiv dekodningsalgoritme for C_i. En sådan algoritme noteres med DC_i, og kan rette t_i fejl. Når der ikke er et kodeord $\mathbf{c} \in C_i$ med afstand højst t_i fra cifferteksten $\mathbf{y}_i$ returnerer algoritmen DC_i *„failure“*.

Nu forklares de forskellige trin i algoritme 12. Som input tager algoritmen en ciffertekst $\mathbf{y} = \mathbf{c} + \mathbf{e}$, hvor $w(\mathbf{e}) \leq t$ samt en matrixprodukt-kode bestående af de indlejrede koder $C_1 \supset C_2 \supset \ldots \supset C_s$ og matricen A, der er ikke-singulær ved søjler. Det kræves desuden, at DC_i haves for C_i, hvor $i = 1, \ldots, s$. Output bliver efter gennemløb af algoritme 12 et kodeord $\mathbf{c}$ i matrixprodukt-koden $C = \begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix} \cdot A$.

Det vides fra ligning (4.4), at kodeord i en matrixprodukt-kode er på formen

$$\mathbf{c} = \left[\sum_{j=1}^{s} a_{j,1}\mathbf{c}_j, \sum_{j=1}^{s} a_{j,2}\mathbf{c}_j, \ldots, \sum_{j=1}^{s} a_{j,\ell}\mathbf{c}_j \right],$$

hvormed et kodeord kan deles op i ℓ dele, hver af længde n. På tilsvarende måde kan fejlvektoren $\mathbf{e}$ opdeles i ℓ dele $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_\ell$. Dermed kan cifferteksten også opdeles i ℓ dele, og den i'te blok af $\mathbf{y}$ noteres som

$$\mathbf{y}_i = \sum_{j=1}^{s} a_{j,i}\mathbf{c}_j + \mathbf{e}_i,$$

hvor $i = 1, \ldots, \ell$.

Det er i punkt 2 nok at vælge en ordnet indeksmængde bestående af s elementer, $\{i_1, \ldots, i_s\} \subset \{1, \ldots, \ell\}$. Dette følger, da matricen A er ikke-singulær ved søjler, og derfor har s lineært uafhængige søjler. Dermed gælder, at når ℓ er større end s, så kan s lineært uafhængige vilkårlige søjler danne de resterende $\ell - s$ søjler ved linearkombination. Det antages yderligere, at $w(\mathbf{e}_{i_j}) \leq t_j$ for alle $j \in \{1, \ldots, s\}$.

Da der er tale om indlejrede koder vil alle blokke af $\mathbf{c}$ være et kodeord i C_1. Derfor kan blok i_1 dekodes ved brug af DC_1. Dette gøres i punkt $4-5$. Hvis DC_1 returnerer *"failure"* vælges en anden indeksmængde i punkt 2. Når DC_1 dekoder effektivt bliver output af denne $\mathbf{y}^2_{i_1}$, der altså er fejlfri. For at det er muligt at dekode $i_2, \ldots, i_s$ ved brug af $DC_2, \ldots, DC_s$ kræves, at disse ikke indeholder et kodeord $\mathbf{c}_1$, der er indeholdt i C_1 men ikke i $C_2, \ldots, C_s$. Derfor ønskes det, at fjerne $\mathbf{c}_1$ fra de resterende $i_2, \ldots, i_s$ blokke. Da $\mathbf{c_1}$ ikke kendes, er dette ikke umiddelbart, men kan gøres ved at opdatere $\mathbf{y}_{i_k}$ for $k = j+1, \ldots, s$ på følgende måde.

$$\begin{aligned}
\mathbf{y}^2_{i_k} &= \mathbf{y}_{i_k} - \frac{a_{1,i_k}}{a_{1,i_1}}(\mathbf{y}^2_{i_1}) \\
&= \left(\sum_{j=1}^{s} a_{j,i_k}\mathbf{c}_j + \mathbf{e}_{i_k} \right) - \frac{a_{1,i_k}}{a_{1,i_1}} \left(\sum_{j=1}^{s} a_{j,i_1}\mathbf{c}_j \right) \\
&= (a_{1,i_k}\mathbf{c}_1 + \ldots + a_{s,i_k}\mathbf{c}_s + \mathbf{e}_{i_k}) - \frac{a_{1,i_k}}{a_{1,i_1}}(a_{1,i_1}\mathbf{c}_1 + \ldots + a_{s,i_1}\mathbf{c}_s) \\
&= \sum_{j=2}^{s} \left(a_{j,i_k} - \frac{a_{1,i_k}}{a_{1,i_1}} a_{j,i_1} \right) \mathbf{c}_j + \mathbf{e}_{i_k}. \qquad (4.10)
\end{aligned}$$

Det ses altså, at de led, der indeholder $\mathbf{c}_1$ går ud med hinanden, hvormed $\mathbf{c}_1$ ikke længere indgår i blok $i_2, \ldots, i_s$. Dette sker i punkt $9-10$ i algoritme 12. Ved opdatering af blokkene er det vigtigt, at $a_{1,i_1} \neq 0$, da dette medfører, at der ikke bliver en division med nul. Dette sker ikke, da A er ikke-singulær ved søjler, hvilket medfører at samtlige indgange i første

række er forskellige fra nul. I punkt $11-12$ opdateres matricen A, hvilket gøres for at sikre, at der ikke divideres med nul i de efterfølgende gennemløb af punkt $9-10$. Dette gøres ved at garantere, at indgang $a_{j,i_j} \neq 0$ for $j = 2, \ldots, s$, hvilket gøres på følgende måde.

$$\text{søjle}_{i_k}(A) = \text{søjle}_{i_k}(A) - \frac{a_{j,i_k}}{a_{j,i_j}} \text{søjle}_{i_j}(A)$$

Dermed bliver matricen $A(i_1, \ldots, i_k)$ en triangulær matrix, og da A er ikke-singulær ved søjler, må $A(i_1, \ldots, i_k)$ være ikke-singulær jævnfør definition 4.1.9. Altså er diagonalindgangene forskellige fra nul, og det er netop disse, der skal divideres med ved efterfølgende gennemløb af punkt $9-10$. Den opdaterede A noteres ved A^δ og indgangene heri ved a^δ_{j,i_k}, hvor $\delta = 1, \ldots, s$ og $A^1 = A$. Dermed kan summen i ligning (4.10) skrives som

$$\mathbf{y}^2_{i_k} = \sum_{j=2}^{s} a^2_{j,i_k} \mathbf{c}_j + \mathbf{e}_{i_k}. \tag{4.11}$$

Proceduren for punkt $4-13$ er tilsvarende i de resterende gennemløb af algoritme 12, og ligning (4.11) skrives generelt som

$$\mathbf{y}^\delta_{i_k} = \sum_{j=\delta}^{s} a^\delta_{j,i_k} \mathbf{c}_j + \mathbf{e}_{i_k}.$$

I punkt $14-15$ opnås kodeordet $\mathbf{c}$ ud fra de opdaterede $\mathbf{y}$'er. Først dannes fejlvektorerne $\mathbf{e}_{i_j}$ for $j = 1, \ldots, s$ ved at subtraktere output i DC_j fra input i DC_j, svarende til $\mathbf{y} - \mathbf{c}$, hvor $\mathbf{y}$ er opdateret. Disse fejlvektorer benyttes til at opnå de oprindelige s blokke af kodeordet $\mathbf{c}$, og $\mathbf{c}_j = \mathbf{y}_j - \mathbf{e}_{i_j}$ for $j = 1, \ldots, s$. For at opnå det oprindelige kodeord $\mathbf{c}$ laves følgende beregning.

$$\mathbf{c} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \ldots & \mathbf{c}_s \end{bmatrix} \cdot A(i_1, \ldots, i_s)^{-1} A$$

Til sidst undersøges, hvor mange fejl, der er rettet. I det tilfælde, hvor der er rettet flere end t fejl, har dekodningsalgoritmen ikke dekodet rigtigt, og en ny indeksmængde skal derfor vælges i punkt 2. Når der derimod er rettet højst t fejl, og det fundne $\mathbf{c}$ ligger i koden, så er dekodningen effektiv, og det rigtige kodeord er fundet. Algoritme 12 terminerer effektivt på grund af følgende sætning

4.2.3 Sætning:
Lad C være en matrixprodukt-kode $\begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix} \cdot A$, hvor delkoderne $C_1, C_2, \ldots, C_s$ er indlejrede på følgende måde,

$$C_1 \supset C_2 \supset \cdots \supset C_s$$

og matricen A er ikke-singulær ved søjler. Lad yderligere $\mathbf{e} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \ldots & \mathbf{e}_\ell \end{bmatrix} \in \mathbb{F}_q^{n\ell}$ være en fejlvektor, hvorom det gælder, at

$$w(\mathbf{e}) \leq t = \left\lfloor \frac{d(C) - 1}{2} \right\rfloor.$$

Så eksisterer en ordnet delmængde $\{i_1, i_2, \ldots, i_s\} \subset \{1, 2, \ldots, \ell\}$, som opfylder

$$w(\mathbf{e}_{i_j}) \leq t_j \ \forall \ j \in \{1, 2, \ldots, s\},$$

og derfor virker algoritme 12 efter hensigten.

Bevis:
Først vises, at der eksisterer et i_1, sådan at $w(\mathbf{e}_{i_1}) \leq t_1$, hvilket gøres ved modstrid. Antag, at der ikke er et $i_1 \in \{1, 2, \ldots, \ell\}$, som har $w(\mathbf{e}_{i_1}) \leq t_1$, hvilket betyder, at

$$w(\mathbf{e}_i) > \left\lfloor \frac{d_1 - 1}{2} \right\rfloor \ \forall \ i = 1, 2, \ldots, \ell.$$

Dette kan omskrives, så $w(\mathbf{e}_i) \geq \frac{d_1}{2} \ \forall \ i$. Der skal nu ses på fejlvægten af den samlede fejlvektor $\mathbf{e}$. Da alle ℓ del-fejlvektorer har en vægt større end eller lig $\frac{d_1}{2}$, må $\mathbf{e}$ have en fejlvægt, hvor følgende gælder.

$$w(\mathbf{e}) \geq \frac{\ell d_1}{2} > \frac{\ell d_1 - 1}{2} \geq \left\lfloor \frac{\ell d_1 - 1}{2} \right\rfloor$$

Fra ligning (4.8) haves, at $d(C) = \min\{\ell d_1, (\ell - 1)d_2, \ldots, (\ell - s + 1)d_s\}$, og dermed må $w(\mathbf{e}) > \left\lfloor \frac{d(C)-1}{2} \right\rfloor$, men dette er i modstrid med antagelsen, at $w(\mathbf{e}) \leq t$. Det er nu vist, at der eksisterer i_1, sådan at $w(\mathbf{e}_{i_1}) \leq t_1$.

Det antages, at egenskaben gælder for en delmængde $\{i_1, i_2, \ldots, i_{j-1}\} \subset \{1, 2, \ldots, \ell\}$, som har kardinalitet $j - 1 < s$.

Nu vises, at egenskaben også gælder for en delmængde med kardinalitet j. Antag, at der ikke eksisterer et i_j, hvor $w(\mathbf{e}_{i_j}) \leq t_j$. Det vil sige

$$w(\mathbf{e}_i) > \left\lfloor \frac{d_j - 1}{2} \right\rfloor \ \forall \ i = \{1, 2, \ldots, \ell\} \backslash \{i_1, i_2, \ldots, i_{j-1}\},$$

hvilket betyder at $w(\mathbf{e}_i) \geq \frac{d_j}{2} \ \forall \ i = \{1, 2, \ldots, \ell\} \backslash \{i_1, i_2, \ldots, i_{j-1}\}$. Den samlede fejlvektor $\mathbf{e}$ må derfor have en fejlvægt bestående af summen af vægten af alle foregående del-fejlvektorer $\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \ldots, \mathbf{e}_{i_{j-1}}$, samt vægten af de $(\ell - (j - 1))$ fejlvektorer $\mathbf{e}_{i_j}$, der er er tilbage. Derfor haves følgende

$$\begin{aligned} w(\mathbf{e}) &\geq \sum_{k=1}^{j-1} w(\mathbf{e}_{i_k}) + \frac{(l - j + 1)d_j}{2} \\ &> \sum_{k=1}^{j-1} w(\mathbf{e}_{i_k}) + \frac{(l - j + 1)d_j - 1}{2} \\ &\geq \left\lfloor \frac{(l - j + 1)d_j - 1}{2} \right\rfloor. \end{aligned} \tag{4.12}$$

Det er ved tredje ulighedstegn muligt at fjerne det første led med fejlvægten af de foregående $\mathbf{e}_{i_j}$, da disse kun kan have en vægt på 0, eller større. Herefter ses igen på ligning (4.8), og det vides fra denne, at $d(C)$ er mindre end eller lig $(l - j + 1)d_j$. Dermed haves fra uligheden i (4.12), at

$$w(\mathbf{e}) > \left\lfloor \frac{d(C) - 1}{2} \right\rfloor,$$

men dette er i modstrid med antagelsen, hvorfor der må eksistere en ordnet delmængde $\{i_1, i_2, \ldots, i_s\} \subset \{1, 2, \ldots, \ell\}$, som opfylder at $w(\mathbf{e}_{i_j}) \leq t_j$. ■

Nu gives et eksempel på brugen af algoritme 12.

4.2.4 Eksempel:

Beregningerne i dette eksempel er lavet i SAGE, og koden står i appendiks B.19. Eksemplet bygger på eksempel 4.2.1 og eksempel 4.2.2. I eksemplet benyttes algoritme 3 og herunder algoritme 12, da der arbejdes med en matrixprodukt-kode. Desuden er delkoderne alle Reed-Solomon-koder, og derfor benyttes algoritmen i appendiks A.1.

Der arbejdes over $\mathbb{F}_{17}$, og fra eksempel 4.2.1 haves den private nøgle bestående af $C_1 = RS_{17}[16, 10]$, $C_2 = RS_{17}[16, 9]$ og $C_3 = RS_{17}[16, 8]$ samt matricerne A, S, P og G. I eksempel 4.2.2 danner Alice cifferteksten

$$\mathbf{y} = [\, 8\ 16\ 14\ 4\ 11\ 14\ 0\ 0\ 16\ 14\ 5\ 11\ 6\ 6\ 1\ 1\ 11\ 16\ 14\ 4\ 14\ 9\ 13\ 1\ 11\ 16\ 1\ 5\ 13\ 3\ 5\ 9\ 7\ 9 \\ 16\ 12\ 11\ 2\ 3\ 2\ 2\ 14\ 13\ 15\ 10\ 4\ 16\ 3\,],$$

og det er netop denne ciffertekst, som Bob i dette eksempel ønsker at dekode. For at dekode skal Bob benytte algoritme 3, og det første han gør er derfor at beregne $\hat{\mathbf{y}}$. Han opnår følgende

$$\hat{\mathbf{y}} = \mathbf{y}P^{-1} = [\, 16\ 4\ 14\ 0\ 14\ 11\ 6\ 1\ 16\ 4\ 9\ 1\ 16\ 5\ 3\ 9\ 9\ 12\ 2\ 2\ 14\ 15\ 4\ 3\ 16\ 10\ 13\ 2\ 3\ 11\ 16\ 7\ 5 \\ 13\ 1\ 11\ 13\ 14\ 14\ 11\ 1\ 6\ 5\ 16\ 0\ 11\ 14\ 8\,].$$

Bob benytter nu algoritme 12 til at dekode $\hat{\mathbf{y}}$ til $\mathbf{c}$. Først opdeles $\hat{\mathbf{y}}$ i $s = 3$ dele hver af længde $n = 16$.

$$\mathbf{y}_1 = \begin{bmatrix} 16 & 4 & 14 & 0 & 14 & 11 & 6 & 1 & 16 & 4 & 9 & 1 & 16 & 5 & 3 & 9 \end{bmatrix}$$

$$\mathbf{y}_2 = \begin{bmatrix} 9 & 12 & 2 & 2 & 14 & 15 & 4 & 3 & 16 & 10 & 13 & 2 & 3 & 11 & 16 & 7 \end{bmatrix}$$

$$\mathbf{y}_3 = \begin{bmatrix} 5 & 13 & 1 & 11 & 13 & 14 & 14 & 11 & 1 & 6 & 5 & 16 & 0 & 11 & 14 & 8 \end{bmatrix}$$

Bob har i algoritme 12 punkt 2 seks mulige valg af indeksmængde

$$\begin{aligned} i_1 &= 1 \quad i_2 = 2 \quad i_3 = 3 \\ i_1 &= 1 \quad i_2 = 3 \quad i_3 = 2 \\ i_1 &= 2 \quad i_2 = 1 \quad i_3 = 3 \\ i_1 &= 2 \quad i_2 = 3 \quad i_3 = 1 \\ i_1 &= 3 \quad i_2 = 1 \quad i_3 = 2 \\ i_1 &= 3 \quad i_2 = 2 \quad i_3 = 1, \end{aligned}$$

og han vælger nu indeksmængden $i_1 = 1$, $i_2 = 2$ og $i_3 = 3$. Han benytter DC_1, der er bygger på algoritmen i appendiks A.1, til at dekode $\mathbf{y}_{i_1}$. Da $Q_0(x)$ deler $Q_1(x)$ opnår Bob

følgende polynomium

$$f(x) = 9x^9 + 6x^8 + 8x^7 + 6x^6 + 16x^5 + 8x^4 + 11x^3 + 3x^2 + 7x + 10,$$

og $\mathbf{y}_1$ dekodes til

$$\mathbf{y}_1^2 = \left[\begin{array}{cccccccccccccccc} 16 & 4 & 14 & 14 & 14 & 11 & 6 & 1 & 16 & 4 & 9 & 1 & 16 & 5 & 3 & 9 \end{array} \right].$$

Nu gennemløber Bob for-løkken i punkt $9-12$ i algoritme 12, og han opdaterer dermed $\mathbf{y}_{i_2}$, $\mathbf{y}_{i_3}$ og A. Første gennemløb er for $k=2$, og her opdateres $\mathbf{y}_2$ til

$$\begin{aligned} \mathbf{y}_2^2 &= \mathbf{y}_2 - \frac{a_{1,i_2}}{a_{1,i_1}}\mathbf{y}_1^2 = \mathbf{y}_2 - \frac{5}{7}\mathbf{y}_1^2 \\ &= \left[\begin{array}{cccccccccccccccc} 0 & 14 & 9 & 9 & 4 & 12 & 7 & 12 & 7 & 12 & 9 & 11 & 11 & 5 & 9 & 3 \end{array} \right]. \end{aligned}$$

Derudover opdateres søjle $i_2 = 2$ i A

$$\text{søjle}_2^2(A) = \text{søjle}_2(A) - \frac{a_{1,i_2}}{a_{1,i_1}}\text{søjle}_1(A) = \begin{bmatrix} 5 \\ 10 \\ 0 \end{bmatrix} - \frac{5}{7}\begin{bmatrix} 7 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 10 \\ 0 \end{bmatrix}.$$

Tilsvarende beregninger laves for $k=3$, og Bob opnår følgende opdateringer. Vektoren $\mathbf{y}_3$ opdateres til

$$\mathbf{y}_3^2 = \left[\begin{array}{cccccccccccccccc} 9 & 14 & 13 & 6 & 8 & 4 & 7 & 7 & 5 & 7 & 3 & 12 & 4 & 8 & 2 & 6 \end{array} \right],$$

og A opdateres til

$$\text{søjle}_3^2(A) = \begin{bmatrix} 0 \\ 9 \\ 3 \end{bmatrix} \qquad A^2 = \begin{bmatrix} 7 & 0 & 0 \\ 0 & 10 & 9 \\ 0 & 0 & 3 \end{bmatrix}.$$

Herefter gennemløber Bob for-løkken i punkt 4 for $j=2$. Derfor benyttes dekodningsalgoritmen DC_2 til at dekode $\mathbf{y}_2^2$. Denne dekodning fejler, da $Q_0(x)$ ikke deler $Q_1(x)$, og DC_2 melder *„failure“*. Bob forsøger derfor med et nyt valg af indeksmængde, og dette gøres, indtil han finder en indeksmængde, der dekoder effektivt. Det viser sig at være tilfældet for indeksmængden $i_1 = 2$, $i_2 = 1$ og $i_3 = 3$. Derfor skal han nu benytte DC_1, der er bygger på algoritmen i appendiks A.1, til at dekode $\mathbf{y}_2$. I DC_1 er $\ell_0 = 12$ og $\ell_1 = 3$, og Bob skal derfor løse det homogene lineære ligningssystem bestående af 16 ligninger med 17 ubekendte, som er på følgende form

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 9 & 9 & 9 & 9 \\
1 & 3 & 9 & 10 & 13 & 5 & 15 & 11 & 16 & 14 & 8 & 7 & 4 & 12 & 2 & 6 & 1 \\
1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 & 1 & 9 & 13 & 15 & 16 & 2 & 1 & 9 & 13 \\
1 & 10 & 15 & 14 & 4 & 6 & 9 & 5 & 16 & 7 & 2 & 3 & 13 & 2 & 3 & 13 & 11 \\
1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 14 & 12 & 3 & 5 \\
1 & 5 & 8 & 6 & 13 & 14 & 2 & 10 & 16 & 12 & 9 & 11 & 4 & 15 & 7 & 1 & 5 \\
1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 & 1 & 15 & 4 & 9 & 16 & 4 & 9 & 16 & 2 \\
1 & 11 & 2 & 5 & 4 & 10 & 8 & 3 & 16 & 6 & 15 & 12 & 13 & 3 & 16 & 6 & 15 \\
1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 \\
1 & 14 & 9 & 7 & 13 & 12 & 15 & 6 & 16 & 3 & 8 & 10 & 4 & 10 & 4 & 5 & 2 \\
1 & 8 & 13 & 2 & 16 & 9 & 4 & 15 & 1 & 8 & 13 & 2 & 16 & 13 & 2 & 16 & 9 \\
1 & 7 & 15 & 3 & 4 & 11 & 9 & 12 & 16 & 10 & 2 & 14 & 13 & 2 & 14 & 13 & 6 \\
1 & 4 & 16 & 13 & 1 & 4 & 16 & 13 & 1 & 4 & 16 & 13 & 1 & 3 & 12 & 14 & 5 \\
1 & 12 & 8 & 11 & 13 & 3 & 2 & 7 & 16 & 5 & 9 & 6 & 4 & 11 & 13 & 3 & 2 \\
1 & 2 & 4 & 8 & 16 & 15 & 13 & 9 & 1 & 2 & 4 & 8 & 16 & 16 & 15 & 13 & 9 \\
1 & 6 & 2 & 12 & 4 & 7 & 8 & 14 & 16 & 11 & 15 & 5 & 13 & 7 & 8 & 14 & 16
\end{bmatrix}
\begin{bmatrix}
Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ Q_{0,3} \\ Q_{0,4} \\ Q_{0,5} \\ Q_{0,6} \\ Q_{0,7} \\ Q_{0,8} \\ Q_{0,9} \\ Q_{0,10} \\ Q_{0,11} \\ Q_{0,12} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ Q_{1,3}
\end{bmatrix} =
\begin{bmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{bmatrix}.$$

Han sætter matricen på reduceret-række-echelon-form

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 12 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 13 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 11 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 5 & 15 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 8 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 4 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 5 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 8 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix},$$

og vælger $Q_{1,3}$ som den frie variabel med værdien 1. Dermed opnår han

$$\begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ Q_{0,3} \\ Q_{0,4} \\ Q_{0,5} \\ Q_{0,6} \\ Q_{0,7} \\ Q_{0,8} \\ Q_{0,9} \\ Q_{0,10} \\ Q_{0,11} \\ Q_{0,12} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ Q_{1,3} \end{bmatrix} = \begin{bmatrix} 16 \\ 5 \\ 4 \\ 15 \\ 16 \\ 6 \\ 14 \\ 8 \\ 13 \\ 2 \\ 16 \\ 11 \\ 13 \\ 11 \\ 16 \\ 0 \end{bmatrix},$$

der giver polynomierne

$Q_0(x) = 16 + 5x + 4x^2 + 15x^3 + 16x^4 + 6x^5 + 14x^6 + 8x^7 + 13x^8 + 2x^9 + 16x^{10} + 11x^{11} + 13x^{12}$
$Q_1(x) = 11 + 16x + x^3$.

Derfor opnås

$$f(x) = -\frac{Q_0(x)}{Q_1(x)} = 14 + 7x + 8x^2 + 12x^3 + 16x^4 + 11x^5 + 11x^6 + 5x^7 + 6x^8 + 4x^9$$

og ved evaluering af x_i'erne i $f(x)$ bliver $\mathbf{y}_2$ dekodet til

$$\mathbf{y}_2^2 = \begin{bmatrix} 9 & 12 & 2 & 2 & 14 & 15 & 4 & 3 & 16 & 10 & 13 & 2 & 3 & 11 & 14 & 9 \end{bmatrix}.$$

Nu gennemløber Bob for-løkken i punkt $9-12$ i algoritme 12, og han opdaterer dermed $\mathbf{y}_{i_2} = \mathbf{y}_1$, $\mathbf{y}_{i_3} = \mathbf{y}_3$ og A. Første gennemløb er for $k = 2$, og her opdateres $\mathbf{y}_1$ til

$$\begin{aligned} \mathbf{y}_1^2 &= \mathbf{y}_1 - \frac{a_{1,i_2}}{a_{1,i_1}}\mathbf{y}_2^2 \\ &= \mathbf{y}_1 - \frac{7}{5}\mathbf{y}_2^2 \\ &= \begin{bmatrix} 0 & 11 & 1 & 4 & 8 & 7 & 14 & 7 & 14 & 7 & 1 & 5 & 5 & 10 & 14 & 10 \end{bmatrix}. \end{aligned}$$

Derudover opdateres søjle $i_2 = 1$ i A

$$\text{søjle}_1^2(A) = \text{søjle}_1(A) - \frac{a_{1,i_2}}{a_{1,i_1}}\text{søjle}_2(A) = \begin{bmatrix} 7 \\ 0 \\ 0 \end{bmatrix} - \frac{7}{5}\begin{bmatrix} 5 \\ 10 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}.$$

Tilsvarende beregninger laves for $k = 3$, og Bob opnår følgende opdateringer,

$$\mathbf{y}_3^2 = \begin{bmatrix} 9 & 7 & 0 & 10 & 6 & 15 & 12 & 1 & 10 & 1 & 7 & 15 & 7 & 14 & 7 & 12 \end{bmatrix}$$

og

$$\text{søjle}_3^2(A) = \begin{bmatrix} 0 \\ 4 \\ 3 \end{bmatrix}.$$

Dermed bliver

$$A^2 = \begin{bmatrix} 0 & 5 & 0 \\ 3 & 10 & 4 \\ 0 & 0 & 3 \end{bmatrix}.$$

Herefter gennemløber Bob for-løkken i punkt 4 for $j = 2$. Derfor benyttes dekodningsalgoritme DC_2 til at dekode $\mathbf{y}_1^2$, og det haves her at $\ell_0 = 12$ og $\ell_1 = 3$, og Bob skal derfor løse det homogene lineære ligningssystem bestående af 16 ligninger med 17 ubekendte, som er på følgende form

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 3 & 9 & 10 & 13 & 5 & 15 & 11 & 16 & 14 & 8 & 7 & 4 & 11 & 16 & 14 & 8 \\
1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 & 1 & 9 & 13 & 15 & 16 & 1 & 9 & 13 & 15 \\
1 & 10 & 15 & 14 & 4 & 6 & 9 & 5 & 16 & 7 & 2 & 3 & 13 & 4 & 6 & 9 & 5 \\
1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 8 & 2 & 9 & 15 \\
1 & 5 & 8 & 6 & 13 & 14 & 2 & 10 & 16 & 12 & 9 & 11 & 4 & 7 & 1 & 5 & 8 \\
1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 & 1 & 15 & 4 & 9 & 16 & 14 & 6 & 5 & 7 \\
1 & 11 & 2 & 5 & 4 & 10 & 8 & 3 & 16 & 6 & 15 & 12 & 13 & 7 & 9 & 14 & 1 \\
1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 14 & 3 & 14 & 3 \\
1 & 14 & 9 & 7 & 13 & 12 & 15 & 6 & 16 & 3 & 8 & 10 & 4 & 7 & 13 & 12 & 15 \\
1 & 8 & 13 & 2 & 16 & 9 & 4 & 15 & 1 & 8 & 13 & 2 & 16 & 1 & 8 & 13 & 2 \\
1 & 7 & 15 & 3 & 4 & 11 & 9 & 12 & 16 & 10 & 2 & 14 & 13 & 5 & 1 & 7 & 15 \\
1 & 4 & 16 & 13 & 1 & 4 & 16 & 13 & 1 & 4 & 16 & 13 & 1 & 5 & 3 & 12 & 14 \\
1 & 12 & 8 & 11 & 13 & 3 & 2 & 7 & 16 & 5 & 9 & 6 & 4 & 10 & 1 & 12 & 8 \\
1 & 2 & 4 & 8 & 16 & 15 & 13 & 9 & 1 & 2 & 4 & 8 & 16 & 14 & 11 & 5 & 10 \\
1 & 6 & 2 & 12 & 4 & 7 & 8 & 14 & 16 & 11 & 15 & 5 & 13 & 10 & 9 & 3 & 1
\end{bmatrix}
\begin{bmatrix}
Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ Q_{0,3} \\ Q_{0,4} \\ Q_{0,5} \\ Q_{0,6} \\ Q_{0,7} \\ Q_{0,8} \\ Q_{0,9} \\ Q_{0,10} \\ Q_{0,11} \\ Q_{0,12} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ Q_{1,3}
\end{bmatrix} =
\begin{bmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{bmatrix}.$$

Han sætter matricen på reduceret-række-echelon-form

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 & 8 & 12 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 8 & 12 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 8 & 6 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 6 & 7 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 14 & 5 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 14 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 & 11 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 6 & 9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 8 & 13 & 7 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 11 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 10 & 15 & 14 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix},$$

og vælger $Q_{1,3}$ som den frie variabel med værdien 1. Dermed opnår han

$$\begin{bmatrix}
Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ Q_{0,3} \\ Q_{0,4} \\ Q_{0,5} \\ Q_{0,6} \\ Q_{0,7} \\ Q_{0,8} \\ Q_{0,9} \\ Q_{0,10} \\ Q_{0,11} \\ Q_{0,12} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ Q_{1,3}
\end{bmatrix} = \begin{bmatrix}
5 \\ 5 \\ 11 \\ 10 \\ 12 \\ 3 \\ 0 \\ 8 \\ 10 \\ 6 \\ 16 \\ 16 \\ 0 \\ 3 \\ 0 \\ 0
\end{bmatrix},$$

der giver polynomierne

$$Q_0(x) = 5 + 5x + 11x^2 + 10x^3 + 12x^4 + 3x^5 + 14x^6 + 8x^7 + 10x^8 + 6x^9 + 16x^{10} + 16x^{11}$$
$$Q_1(x) = 3 + x^3.$$

Dermed opnås

$$f(x) = -\frac{Q_0(x)}{Q_1(x)} = 4 + 4x + 2x^2 + x^3 + 6x^4 + 4x^5 + 11x^6 + x^7 + x^8$$

og ved evaluering af x_i'erne i $f(x)$ bliver $\mathbf{y}_1^2$ dekodet til

$$\mathbf{y}_1^3 = \begin{bmatrix} 0 & 11 & 1 & 1 & 8 & 7 & 14 & 7 & 14 & 7 & 1 & 5 & 5 & 10 & 14 & 10 \end{bmatrix}.$$

Nu gennemløber Bob for-løkken i punkt $9-12$ i algoritme 12, og han opdaterer dermed $\mathbf{y}_3^2$ og A^2. Da $s=3$ skal denne løkke kun gennemløbes for $k=3$, og her opdateres $\mathbf{y}_3^2$ til

$$\begin{aligned}\mathbf{y}_3^3 &= \mathbf{y}_3^2 - \frac{a_{2,i_3}}{a_{2,i_2}}\mathbf{y}_1^3 \\ &= \mathbf{y}_3^2 - \frac{4}{3}\mathbf{y}_2^2 \\ &= \begin{bmatrix} 9 & 15 & 10 & 3 & 1 & 0 & 16 & 3 & 14 & 3 & 0 & 14 & 6 & 12 & 11 & 10 \end{bmatrix}.\end{aligned}$$

Derudover opdateres søjle $i_3 = 3$ i A^2

$$\text{søjle}_3^3(A) = \text{søjle}_3^2(A) - \frac{a_{2,i_3}}{a_{2,i_2}}\text{søjle}_1^2(A) = \begin{bmatrix}0\\4\\3\end{bmatrix} - \frac{4}{3}\begin{bmatrix}0\\3\\0\end{bmatrix} = \begin{bmatrix}0\\0\\3\end{bmatrix},$$

og dermed bliver

$$A^3 = \begin{bmatrix}0 & 5 & 0\\3 & 10 & 0\\0 & 0 & 3\end{bmatrix}.$$

Nu er algoritme 12 punkt 4 gennemløbet med succes for $j = 1, 2$ og Bob skal nu gennemløbe den for $j = 3$. Derfor benyttes dekodningsalgoritme DC_3 til at dekode $\mathbf{y}_3^3$, og det haves her at $\ell_0 = 11$ og $\ell_1 = 4$, og Bob skal derfor løse det homogene lineære ligningssystem bestående af 16 ligninger med 17 ubekendte, som er på følgende form

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 9 & 9 & 9 & 9 & 9\\
1 & 3 & 9 & 10 & 13 & 5 & 15 & 11 & 16 & 14 & 8 & 7 & 15 & 11 & 16 & 14 & 8\\
1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 & 1 & 9 & 13 & 15 & 10 & 5 & 11 & 14 & 7\\
1 & 10 & 15 & 14 & 4 & 6 & 9 & 5 & 16 & 7 & 2 & 3 & 3 & 13 & 11 & 8 & 12\\
1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1\\
1 & 5 & 8 & 6 & 13 & 14 & 2 & 10 & 16 & 12 & 9 & 11 & 0 & 0 & 0 & 0 & 0\\
1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 & 1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 & 1\\
1 & 11 & 2 & 5 & 4 & 10 & 8 & 3 & 16 & 6 & 15 & 12 & 3 & 16 & 6 & 15 & 12\\
1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 14 & 3 & 14 & 3 & 14\\
1 & 14 & 9 & 7 & 13 & 12 & 15 & 6 & 16 & 3 & 8 & 10 & 3 & 8 & 10 & 4 & 5\\
1 & 8 & 13 & 2 & 16 & 9 & 4 & 15 & 1 & 8 & 13 & 2 & 0 & 0 & 0 & 0 & 0\\
1 & 7 & 15 & 3 & 4 & 11 & 9 & 12 & 16 & 10 & 2 & 14 & 14 & 13 & 6 & 8 & 5\\
1 & 4 & 16 & 13 & 1 & 4 & 16 & 13 & 1 & 4 & 16 & 13 & 6 & 7 & 11 & 10 & 6\\
1 & 12 & 8 & 11 & 13 & 3 & 2 & 7 & 16 & 5 & 9 & 6 & 12 & 8 & 11 & 13 & 3\\
1 & 2 & 4 & 8 & 16 & 15 & 13 & 9 & 1 & 2 & 4 & 8 & 11 & 5 & 10 & 3 & 6\\
1 & 6 & 2 & 12 & 4 & 7 & 8 & 14 & 16 & 11 & 15 & 5 & 10 & 9 & 3 & 1 & 6
\end{bmatrix}
\begin{bmatrix}
Q_{0,0}\\Q_{0,1}\\Q_{0,2}\\Q_{0,3}\\Q_{0,4}\\Q_{0,5}\\Q_{0,6}\\Q_{0,7}\\Q_{0,8}\\Q_{0,9}\\Q_{0,10}\\Q_{0,11}\\Q_{1,0}\\Q_{1,1}\\Q_{1,2}\\Q_{1,3}\\Q_{1,4}
\end{bmatrix}
=
\begin{bmatrix}
0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0
\end{bmatrix}.$$

Han sætter matricen på reduceret-række-echelon-form

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 4 & 9 & 16 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 2 & 13 & 8 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 11 & 10 & 14 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 5 & 5 & 5 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 14 & 8 & 16 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 14 & 11 & 2 & 15 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 6 & 10 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 13 & 12 & 7 & 8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 3 & 7 & 7 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 3 & 7 & 7 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 3 & 7 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 15 & 4 & 9 & 16 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix},$$

og vælger $Q_{1,4}$ som den frie variabel med værdien 1. Dermed opnår han

$$\begin{bmatrix}
Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ Q_{0,3} \\ Q_{0,4} \\ Q_{0,5} \\ Q_{0,6} \\ Q_{0,7} \\ Q_{0,8} \\ Q_{0,9} \\ Q_{0,10} \\ Q_{0,11} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ Q_{1,3} \\ Q_{1,4}
\end{bmatrix} = \begin{bmatrix}
1 \\ 9 \\ 3 \\ 12 \\ 1 \\ 2 \\ 13 \\ 9 \\ 0 \\ 10 \\ 10 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1
\end{bmatrix},$$

der giver polynomierne

$$Q_0(x) = 1 + 9x + 3x^2 + 12x^3 + x^4 + 2x^5 + 13x^6 + 9x^7 + 10x^9 + 10x^{10} + 14x^{11}$$
$$Q_1(x) = 1 + x^4.$$

Dermed opnås

$$f(x) = -\frac{Q_0(x)}{Q_1(x)} = 16 + 8x + 14x^2 + 5x^3 + 7x^5 + 7x^6 + 3x^7$$

og ved evaluering af x_i'erne i $f(x)$ bliver $\mathbf{y}_3^4$ dekodet til

$$\mathbf{y}_3^4 = \begin{bmatrix} 9 & 15 & 10 & 3 & 1 & 0 & 9 & 3 & 14 & 3 & 0 & 14 & 6 & 12 & 11 & 10 \end{bmatrix}.$$

Herefter kommer algoritme 12 ikke ind i for-løkken i punkt $9-13$, og det næste Bob gør er at finde kodeordet $\mathbf{c}$. For at gøre dette finder han fejlvektorerene $\mathbf{e}_j$, som kan findes ved at se på forskellen mellem y_{i_j} og denne dekodet.

$$\mathbf{e}_1 = y_2 - y_2^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 15 \end{bmatrix}$$
$$\mathbf{e}_2 = y_1^2 - y_1^3 = \begin{bmatrix} 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{e}_3 = y_3^3 - y_3^4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Det ses, at fejlvægten af $\mathbf{e} \in \mathbb{F}_{17}^{48}$ er 4, hvilket betyder at der netop er rettet t fejl. Nu er det muligt for Bob at finde kodeordet, der er sendt uden fejl, og dette gør han på følgende måde, hvor $\mathbf{c}_{i_j} = \mathbf{y}_{i_j} - \mathbf{e}_j$.

$$\mathbf{c}_{i_1} = \mathbf{y}_2 - \mathbf{e}_1 = \begin{bmatrix} 9 & 12 & 2 & 2 & 14 & 15 & 4 & 3 & 16 & 10 & 13 & 2 & 3 & 11 & 14 & 9 \end{bmatrix}$$
$$\mathbf{c}_{i_2} = \mathbf{y}_1 - \mathbf{e}_2 = \begin{bmatrix} 16 & 4 & 14 & 14 & 14 & 11 & 6 & 1 & 16 & 4 & 9 & 1 & 16 & 5 & 3 & 9 \end{bmatrix}$$
$$\mathbf{c}_{i_3} = \mathbf{y}_3 - \mathbf{e}_3 = \begin{bmatrix} 5 & 13 & 1 & 11 & 13 & 14 & 7 & 11 & 1 & 6 & 5 & 16 & 0 & 11 & 14 & 8 \end{bmatrix}$$

Bob danner nu $\mathbf{c} = \begin{bmatrix} \mathbf{c}_{i_2} & \mathbf{c}_{i_1} & \mathbf{c}_{i_3} \end{bmatrix}$, og da dette er et kodeord i C har han dekodet korrekt. Nu er algoritme 12 gennemløbet, og Bob skal nu videre til punkt 3 i algoritme 3, hvor han skal beregne meddelelsen ud fra outputtet i dekodningsalgoritmen. Det antages dog her, at dekodningsalgoritmen der benyttes har output $\mathbf{m}S$. Dette er ikke tilfældet for algoritme 12, og derfor skal Bob først finde $\mathbf{m}S$ ud fra $\mathbf{c}$. Dette gør han ved at løse ligningssystemet $\mathbf{m}S \cdot G = \mathbf{c}$, og han opnår følgende

$$\mathbf{m}S = [\ 16\ 1\ 15\ 4\ 6\ 12\ 13\ 6\ 13\ 11\ 7\ 7\ 12\ 6\ 2\ 7\ 15\ 6\ 6\ 11\ 14\ 16\ 13\ 0\ 8\ 8\ 1\ ].$$

Bob opnår meddelelsen på følgende måde

$$\mathbf{m} = \mathbf{m}S \cdot S^{-1} = [\ 11\ 15\ 4\ 14\ 9\ 14\ 7\ 0\ 13\ 5\ 4\ 0\ 3\ 1\ 13\ 9\ 12\ 12\ 1\ 0\ 15\ 7\ 0\ 9\ 4\ 1\ 0\ ].$$

Det ses nu, at denne meddelelse netop er, hvad Alice har sendt til Bob i eksempel 4.2.2.◀

Der ses nu på kompleksiteten af algoritme 12. Først betragtes, hvor mange gange det kan risikeres, at algoritmen skal gennemløbes. Antallet af gennemløb er tilsvarende antallet af forskellige måder, hvorpå en mængde af størrelse s kan vælges ud fra en mængde af størrelse ℓ. Ved første valg er der ℓ forskellige muligheder, ved andet er der $\ell - 1$ og så videre. Dermed bliver der

$$\ell(\ell-1)\cdots(\ell-(s-1)) = \frac{\ell!}{(\ell-s)!}$$

forskellige mulige måder, hvorpå delmængden i punkt 2 i algoritme 12 kan dannes. I hvert gennemløb af algoritmen kan det risikeres, at s dekodningsalgoritmer for delkoderne skal gennemløbes. Kompleksiteten af disse dekodningsalgoritmer DC_i noteres $komp(DC_i)$. Derved må kompleksiteten af algoritmen være

$$\frac{\ell!}{(\ell-s)!}(komp(DC_1) + komp(DC_2) + \ldots + komp(DC_s)).$$

Når ℓ er lille, vil det være kompleksiteten af dekodningsalgoritmerne, der dominerer kompleksiteten af algoritme 12, mens det ved et stort ℓ og s kan blive $\frac{\ell!}{(\ell-s)!}$, der dominerer den samlede kompleksitet. Det er dog meget usandsynligt, at algoritmen skal gennemløbes for alle mulige indeksmængder, før den rette er fundet. Derudover kan der også allerede ske en fejl ved dekodningsalgoritmen for C_2, hvormed det ikke bliver nødvendigt at gennemløbe de resterende dekodningsalgoritmer for denne indeksmængde. Derfor er den gennemsnitlige kompleksitet af algoritme 12 mindre.

En anden betragtning er muligheden for at $DC_1, DC_2, \ldots, DC_s$ er listedekodningsalgoritmer. Dette muligør at rette flere fejl, men gør også, at det er muligt at få en liste med kodeord for hver dekodningsalgoritme. Sandsynligheden for at få en liste med kodeord er dog ifølge (Hernando m.fl., 2012) meget lille, når delkoderne er cykliske.

Teorien omkring matrixprodukt-koder samt hvordan denne kodetype kan benyttes i McEliece kryptosystemet er nu gennemgået. I næste kapitel præsenteres en ny version af McEliece kryptosystemet, hvor det ved brug af matrixprodukt-koder og deres struktur er muligt at reducere den offentlige nøglestørrelse.

KAPITEL 5

En ny version af McEliece kryptosystemet

I artiklen *„Shorter keys for code based cryptography“* præsenterer Gaborit konceptet med at reducere den offentlige nøglestørrelse ved brug af strukturen for quasicykliske koder (Gaborit, 2004). I artiklen introduceres derfor en ny version af McEliece kryptosystemet, hvilket er behandlet i afsnit 3.4.

Tankegangen bag Gaborits version er i særdeleshed interessant i dette projekt, da noget tilsvarende kan gøres for matrixprodukt-koder. Som tidligere nævnt er en generatormatrix for matrixprodukt-koder på formen

$$G = \begin{bmatrix} a_{1,1}G_1 & a_{1,2}G_1 & \dots & a_{1,\ell}G_1 \\ a_{2,1}G_2 & a_{2,2}G_2 & \dots & a_{2,\ell}G_2 \\ \vdots & \vdots & \vdots & \vdots \\ a_{s,1}G_s & a_{s,2}G_s & \dots & a_{s,\ell}G_s \end{bmatrix},$$

hvor G_i er generatormatrix til koden C_i for $i = 1, \dots, s$. Det bemærkes, at generatormatricen for matrixrodukt-koden er dannet af delmatricer, ligesom det er tilfældet for quasicykliske koder. Den nye version, der nu dannes, er ud fra samme tankegang som i Gaborits version, og dermed foreslås det, at det er nok at benytte delgeneratormatricerne og matricen A til generering af den offentlige nøgle. Til generering af den offentlige og private nøgle i den nye version skal en tilsvarende permutation π som i afsnit 3.4 benyttes til G_i'erne. Denne permutation permuterer søjlerne i de enkelte generatormatricer G_i. Da denne permutation er tilsvarende den fra Gaborits version af McEliece kryptosystemet skal det dog nævnes, at det i (Otmani m.fl., 2010) er vist, at Gaborits version ikke er sikker, da permutationen kan findes ud fra kendskaben til at der benyttes delkoder af en kendt BCH-kode. Derfor er det heller ikke sikkert, at denne nye version er sikker, da det er samme tankegang den nye version er genereret ud fra, dog benyttes ikke nødvendigvis BCH-koder.

I det oprindelige McEliece kyptosystem skal generatormatricen G ved nøglegenerering både multipliceres med en invertibel matrix S og en permutationsmatrix P. Det overvejes derfor, om sikkerheden af den nye version af McEliece kryptosystemet kan gøres mere sikker ved samme fremgangsmåde. Da der arbejdes med matrixprodukt-koder, er det dog ikke muligt at multiplicere generatormatricerne G_i med samme S, da

generatormatricerne ikke nødvendigvis har samme rang. Det er yderligere nødvendigt at have en samlet invertibel matrix S til McEliece dekodningen. Dette skyldes, at sidste punkt i dekodningsalgoritmen for McEliece kryptosystemet, algoritme 3, er at finde meddelelsen $\mathbf{m}$ ud fra $\mathbf{m}S$. Dette er dog ikke muligt, når det ønskes at sende generatormatricerene G_i hver for sig, og på den måde reducere den offentlige nøgle størrelse. En idé kan derfor være at danne en matrix bestående af invertible matricer S_i i diagonalen og nulindgange både over og under. Problemet er bare, at det ikke kan garanteres, at en sådan blokmatrix er invertibel, selvom de enkelte S_i'er er det. Derfor vil generatormatricerne G_i i den nye version af McEiece kryptosystemet kun blive multipliceret med en permutationsmatrix π.

5.1 Algoritmer til ny version af McEliece kryptosystemet

I dette afsnit gives algoritmer til nøglegenerering, indkoding og dekodning i den nye version af McEliece kryptosystemet med matrixprodukt-koder. Ved nøglegenerering vælges indlejrede delkoder og en ikke-singulær ved søjler matrix A, da det er for sådanne koder, at en effektiv dekodningsalgoritme kendes, se eventuelt algoritme 12. I det tilfælde, hvor der kendes en effektiv dekodningsalgoritme unden disse betingelser, kan disse undlades i algoritme 13. Herefter skal generatormatricerne for disse delkoder dannes, og en permutationsmatrix Π skal konstrueres. Denne permutationsmatrix skal have samme egenskaber som beskrevet ved Gaborits version, se ligning (3.8). Dog skal permutationen her være ℓ sæt af n koordinater, hvilket svarer til, at søjlerne i hvert G_i permuteres. En generatormatrix for en matrixprodukt-kode er jævnfør ligning (4.5) på formen

$$
\begin{aligned}
G &= \begin{bmatrix} a_{1,1}G_1 & a_{1,2}G_1 & \dots & a_{1,\ell}G_1 \\ a_{2,1}G_2 & a_{2,2}G_2 & \dots & a_{2,\ell}G_2 \\ \vdots & \vdots & \vdots & \vdots \\ a_{s,1}G_s & a_{s,2}G_s & \dots & a_{s,\ell}G_s \end{bmatrix} \\
&= \begin{bmatrix} G_1 & G_1 & \dots & G_1 \\ G_2 & G_2 & \dots & G_2 \\ \vdots & \vdots & \vdots & \vdots \\ G_s & G_s & \dots & G_s \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,\ell} \\ a_{2,1} & a_{2,2} & \dots & a_{2,\ell} \\ \vdots & \vdots & \vdots & \vdots \\ a_{s,1} & a_{s,2} & \dots & a_{s,\ell} \end{bmatrix}.
\end{aligned}
$$

Lad

$$
\widetilde{G} = \begin{bmatrix} G_1 & G_1 & \cdots & G_1 \\ G_2 & G_2 & \cdots & G_2 \\ \vdots & \vdots & \cdots & \vdots \\ G_s & G_s & \cdots & G_s \end{bmatrix} \tag{5.1}
$$

så er $G = \widetilde{G}A$. Nu skal matricen

$$
\widetilde{G}' = \widetilde{G}\Pi = \begin{bmatrix} G_1\pi & G_1\pi & \dots & G_1\pi \\ G_2\pi & G_2\pi & \dots & G_2\pi \\ \vdots & \vdots & \vdots & \vdots \\ G_s\pi & G_s\pi & \dots & G_s\pi \end{bmatrix},
$$

dannes, og de n første søjler udgør sammen med matricen A og parametren t den offentlige nøgle. Bemærk, at de n søjler svarer til $G_1\pi, G_2\pi, \ldots, G_s\pi$. Den private nøgle er permutationsmatricen Π samt en effektiv dekodningsalgoritme for C. Denne form for nøglegenerering er beskrevet i følgende algoritme.

Algoritme 13 Nøglegenerering i den nye version

1: Vælg s indlejrede delkoder $C_1 \supset C_2 \supset \ldots \supset C_s$ samt en ikke-sigulær ved søjler matrix A så $C = \begin{bmatrix} C_1 & C_2 & \cdots & C_s \end{bmatrix} \cdot A$ har $W_{q,n,d,t} \geq 2^\kappa$.
2: Bestem s tilfældige generatormatricer $G_1, G_2, \ldots, G_s$ for henholdsvis $C_1, C_2, \ldots, C_s$.
3: Bestem en permutationsmatrix Π som beskrevet ovenfor.
4: Beregn $\widetilde{G}' = \widetilde{G}\Pi$, hvor $\widetilde{G}$ er som i ligning (5.1).
5: **return** Den offentlige nøgle (matricerne $(G_1\pi, G_2\pi, \ldots, G_s\pi), A, t$) og den private nøgle ($\Pi$, dekodningsalgoritme for C).

Når der i den nye version skal indkodes foregår det efter algoritme 14. Første trin i algoritmen er at danne generatormatricen

$$G' = \begin{bmatrix} a_{1,1}G_1\pi & a_{1,2}G_1\pi & \ldots & a_{1,\ell}G_1\pi \\ a_{2,1}G_2\pi & a_{2,2}G_2\pi & \ldots & a_{2,\ell}G_2\pi \\ \vdots & \vdots & \vdots & \vdots \\ a_{s,1}G_s\pi & a_{s,2}G_s\pi & \ldots & a_{s,\ell}G_s\pi \end{bmatrix} \tag{5.2}$$

ud fra den offentlige nøgle. Derefter indkodes meddelelsen ved $\mathbf{m}G'$. En fejlvektor $\mathbf{e}$ med vægt w, hvor $w \leq t$, og længde $n\ell$ vælges, og cifferteksten $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$ dannes. I det tilfælde, hvor algoritme 12 benyttes, er det vigtigt, at fejlene fordeles, så der ikke er en delfejlvektor $\mathbf{e}_i$ med vægt større end t_i. Dette skyldes, at det i et sådant tilfælde ikke kan garanteres, at det er muligt at rette fejlene og dermed at dekode effektivt.

Algoritme 14 Indkodning i den nye version

Input: En meddelelse $\mathbf{m} \in \mathbb{F}_q^k$, den offentlige nøgle, matricerne $G_1\pi, G_2\pi, \ldots, G_s\pi$, matricen A samt parameteren t.
Output: En cifferskrift $\mathbf{y} \in \mathbb{F}_q^{n\ell}$.

1: Dan generatormatricen G' som beskrevet i ligning (5.2) ud fra $G_1\pi, G_2\pi, \ldots, G_s\pi$ samt matricen A.
2: Beregn $\mathbf{m}G'$.
3: Bestem en tilfældig vektor $\mathbf{e}$ med længde $n\ell$ og vægt w og beregn $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$.
4: **return** Cifferteksten $\mathbf{y}$.

Ved dekodning af den sendte ciffertekst er første trin at danne G' ud fra den offentlige nøgle. Herefter benyttes den private nøgle. Matricen Π benyttes til at beregne $\mathbf{y}' = \mathbf{y}\Pi^{-1}$, og dekodningsalgoritmen for C benyttes til at dekode $\mathbf{y}'$ til $\mathbf{m}$.

Den nye version af McEliece kryptosystemet, der nu er præsenteret, har som tidligere nævnt til formål at reducere den offentlige nøglestørrelse. Dette opnås, da det blot er nødvendigt at sende delmatricerne $G_1\pi, G_2\pi, \ldots, G_s\pi$ samt den ikke-singulær ved søjler

Algoritme 15 Dekodning i den nye version

Input: En vektor $\mathbf{y} = \mathbf{m}G' + \mathbf{e} \in \mathbb{F}_q^{n\ell}$ og den private nøgle (Π, dekodningsalgoritme for C).

Output: Meddelelsen $\mathbf{m}$.

1: Dan generatormatricen G' som beskrevet i ligning (5.2) ud fra $G_1\pi, G_2\pi, \ldots, G_s\pi$ samt matricen A.
2: Beregn $\mathbf{y}' = \mathbf{y}\Pi^{-1} = \mathbf{m}G'\Pi^{-1} + \mathbf{e}\Pi^{-1}$
3: Benyt dekodningsalgoritmen for C på $\mathbf{y}'$ til at finde $\mathbf{m}$.
4: **return** Meddelelsen $\mathbf{m}$.

matrix A fremfor hele G. I en matrixprodukt-kode vil generatormatricen G være en $(k_1 + k_2 + \ldots + k_s) \times n\ell$ matrix, hvilket svarer til, at der sendes $(k_1 + k_2 + \ldots + k_s)n\ell$ bits. I den nye version sendes matricer med følgende størrelser $k_1 \times n$, $k_2 \times n$, …, $k_s \times n$ samt $s \times \ell$. Dette giver $(k_1 + k_2 + \ldots + k_s)n + s\ell$ bits, der skal sendes. For at den offentlige nøglestørrelse er den samme eller reduceret skal følgende ulighed være opfyldt

$$(k_1 + k_2 + \ldots + k_s)n + s\ell \leq (k_1 + k_2 + \ldots + k_s)n\ell.$$

Når denne ulighed er opfyldt ved lighed er den offentlige nøglestørrelse uændret, ellers er den reduceret. Dette svarer til, at

$$s \leq \frac{(k_1 + k_2 + \ldots + k_s)n(\ell - 1)}{\ell}.$$

Når sikkerheden af denne nye version af McEliece kryptosystemet betragtes, skal det overvejes hvilken type af koder, der benyttes. Da Gaborits version af kryptosystemet kan brydes ved at finde permutationen Π og dermed strukturen af koden, betyder dette ikke nødvendigvis, at den nye version der her er præsenteret kan brydes. I Gaborits version er det et krav, at koderne er quasicykliske. Dette krav haves ikke i den nye version, hvorfor det er muligt at overveje hvilke kodetyper, der er sikre at benytte. Intuitivt virker irreducible binære Goppa-koder som et godt valg, da de endnu ikke er brudt for det oprindelige McEliece kryptosystem. Fordelen ved at vælge Goppa-koder som delkoder i en matrixprodukt-kode fremfor blot at arbejde med Goppa-koder i sig selv er, at den offentlige nøglestørrelse kan reduceres. En anden overvejelse er, at længden af delkoderne og dermed den samlede matrixprodukt-kode måske ikke nødvendigvis behøver at være så stor. Intuitivt vil dette være tilfældet, da konstruktionen af en matrixprodukt-kode gør det sværere at finde strukturen, og derfor er længden af koden ikke helt så afgørende for sikkerheden, som det for eksempel er tilfældet, når der abejdes med Goppa-koder.

Afrunding

I dette projekt har vi gennemgået den grundlæggende teori om McEliece kryptosystemet. Ud fra denne har vi arbejdet os frem mod en ny version, der reducerer den ellers store offentige nøgle. For at nå hertil har vi samlet en række relevante resultater, hovedsageligt fra artikler. Vi har gennemgået resultaterne i denne litteratur og tilføjet detaljer. Dermed er teorien bevet samlet, forklaret og beskrevet ved en ensartet notation.

Før en ny version af McEliece kryptosystemet kan dannes er det nødvendigt at have en grundlæggende viden herom. Derfor er McEliece kryptosystemet i dets oprindelige form først beskrevet. Herefter er teorien om Goppa-koder også blevet gennemgået, da det netop er denne type koder, der oprindeligt blev benyttet. I projektet er også præsenteret to andre udgaver af McEliece kryptosystemet, Niederreiters og Gaborits. I Niederreiter kryptosystemet benyttes paritetstjekmatricen i stedet for generatormatricen, og dermed reduceres den offentlige nøglestørrelse. Da de to kryptosystemer er ækvivalente, har vi valgt kun at fokusere på McElieces kryptosystem, da dette er det oprindelige.

Der er to typer af angreb, meddelelsesangreb og nøgleangreb. I projektet er den bagvedliggende teori om disse gennemgået, og det er ud fra simple eksempler blevet illustreret, at begge typer af angreb bliver meget beregningsteknisk svære for selv koder med små parametre. Det er også vist, at der eksisterer utrolig mange mulige Goppapolynomier for koder med samme parametre. Netop dette er grunden til at denne type stadig ikke er brudt.

I projektet er der også arbejdet med quasicykliske koder, da deres struktur er særligt interessant i forhold til at reducere den offentlige nøglestørrelse. Det er i særdeleshed måden hvorpå en matrix, der genererer hele koden, kan dannes, der er interessant. I Gaborits version af McEliece kryptosystemet er det også denne form for generatormatrix, der benyttes. Ved brug af denne reducerer Gaborit den offentlige nøglestørrelse i kryptosystemet. Det er dog bemærket, at Gaborits version af kryptosystemet er blevet brudt, da han bruger delkoder af én kendt BCH-kode. Alligevel er det en interessant betragtning, at generatormatricen kan dannes af delmatricer, og det er denne betragtning, vi benytter til den nye version.

Idéen med den nye version opstår ud fra teorien om matrixprodukt-koder. Det er vist, at en matrixprodukt-kodes generatormatrix også kan dannes ud fra delmatricer, hvor disse delmatricer er generatormatricer for delkoderne i matrixprodukt-koden. Da det ved McEliece kryptosystemet er nødvendigt at have en effektiv dekodningsalgoritme

for den type kode, der benyttes, er en dekodningsalgoritme for matrixprodukt-koder også præsenteret. Denne algoritme har dog nogle restriktioner, hvilket er, at det er nødvendigt at delkoderne er indlejrede, samt at matricen A er ikke-singulær ved søjler. Ved brug af dette er det nu muligt at lave en ny version af McEliece kryptosystemet. I vores version er det, ligesom ved Gaborits version, kun muligt at benytte en permutationsmatrix og altså ikke yderligere benytte en singulær matrix ved nøglegenerering. Vi har dog antaget, at da vi ikke skal benytte cykliske delkoder samt en kendt kodetype, vil versionen ikke have samme sikkerhedsbrist som Gaborits version. Der er altså præsenteret en ny version af McEliece kryptosystemet med en reduceret offentlig nøglestørrelse, der ikke umiddelbart kan brydes.

Perspektivering

I dette projekt har vi fremlagt en ny version af McEliece kryptosystemet, hvor der benyttes matrixprodukt-koder. Det kan i fremtidigt arbejde være interessant at undersøge denne version yderligere. Dette kan blandt andet være at lave en grundigere sikkerhedsanalyse, hvormed de antagelser, der er foretaget i dette projekt kan blive be- eller afkræftet. Hvis det viser sig, at den nye version af kryptosystemet ikke er sikker kan en udvidelse foreslået af Berger til Gaborits version også være relevant at undersøge. Denne udvidelse består i at benytte subfield subkoder til Gaborits version af kryptosystemet (Berger m.fl., 2009). Da subfield subkoder er en af de kodetyper, der stadig er sikre, vil sikkerheden øges. En tilsvarende udvidelse kan tænkes at være relevant for matrixprodukt-koder, hvor delkoderne dannes som subfield subkoder.

En anden indgangsvinkel til fremtidigt arbejde er at betragte dekodningsalgoritmen til matrixprodukt-koder. Den algoritme, der i dette projekt er benyttet, er begrænset af at delkoderne skal være indlejrede samt at matricen A skal være ikke-singulær ved søjler. Det er interessant at undersøge, om der kan udvikles en dekodningsalgoritme for matrixprodukt-koder, hvor disse restriktioner ikke er nødvendige.

Bibliografi

Adams, **William W. og Philippe Loustaunau** (2000). *An Introduction to Gröbner Bases*. 3. udg. American Mathematical Society. ISBN: 978-0-8218-3804-4.

Bauer, **Craig P.** (2013). *Secret history*. Discrete Mathematics and its Applications (Boca Raton). The story of cryptology. CRC Press, Boca Raton, FL, s. xxv+594. ISBN: 978-1-4665-6186-1.

Berger, **Thierry P. og Pierre Loidreau** (2005). „How to mask the structure of codes for a cryptographic use“. I: *Des. Codes Cryptogr.* 35.1, s. 63–79.

Berger, **Thierry P.** m.fl. (2009). „Reducing key length of the McEliece cryptosystem“. I: *Progress in cryptology—AFRICACRYPT 2009*. Bd. 5580. Lecture Notes in Comput. Sci. Springer, Berlin, s. 77–97.

Bernstein, **Daniel J.** (2009). „Introduction to post-quantum cryptography“. I: *Post-quantum cryptography*. Springer, Berlin, s. 1–14.

Blackmore, **Tim og Graham H. Norton** (2001). „Matrix-product codes over $\mathbb{F}_q$“. I: *Appl. Algebra Engrg. Comm. Comput.* 12.6, s. 477–500.

Chizhov, **Ivan V. og Mikhail A. Borodin** (2013). „The failure of McEliece PKC based on Reed-Muller codes.“ I: *IACR Cryptology ePrint Archive*.

Couvreur, **Alain**, **Irene Marquez Corbella og Ruud Pellikaan** (2014). „A Polynomial Time Attack against Algebraic Geometry Code Based Public Key Cryptosystems“. I: *CoRR* abs/1401.6025.

Cox, **David A.**, **John Little og Donal O'Shea** (2005). *Using Algebraic Geometry*. 2. udg. Springer. ISBN: 978-0-387-20706-3.

Faure, **Cédric og Lorenz Minder** (2008). „Cryptanalysis of the McEliece cryptosystem over hyperelliptic codes“. I: *Eleventh International Workshop on Algebraic and Combinatorial Coding Theory*, s. 99–107.

Gaborit, **Philippe** (2004). „Shorter keys for code-based cryptography“. I: *Proceedings of Workshop on Codes and Cryptography*. Université de Limoges, France: WCC 2005, s. 81–90.

Hall, **Jonathan I.** (2015). *Notes on Coding Theory*. Notesæt. URL: http://www.mth.msu.edu/~jhall/classes/codenotes/Cyclic.pdf.

Hernando, **Fernando**, **Kristine Lally og Diego Ruano** (2009). „Construction and decoding of matrix-product codes from nested codes“. I: *Appl. Algebra Engrg. Comm. Comput.* 20.5-6, s. 497–507.

Hernando, **Fernando**, **Tom Høholdt og Diego Ruano** (2012). „List decoding of matrix-product codes from nested codes: an application to quasi-cyclic codes“. I: *Adv. Math. Commun.* 6.3, s. 259–272.

Huffman, **William Cary og Vera Pless** (2003). *Fundamentals of Error-Correcting Codes*. Cambridge University Press. ISBN: 978-0-521-78280-7.

Janwa, **Heeralal og Oscar Moreno** (1996). „McEliece public key cryptosystems using algebraic-geometric codes“. I: *Designs, Codes and Cryptography* 8, s. 293–307.

Justesen, **Jørn og Tom Høholdt** (2004). *A course in error-correcting codes*. EMS Textbooks in Mathematics. European Mathematical Society (EMS), Zürich. ISBN: 3-03719-001-9.

Klein, **Philip N.** (2014). *A Cryptography Primer: Secrets and Promises*. Cambridge University Press. ISBN: 9781139916103.

Lally, **Christine** (2000). „Application of the Theory of Gröbner Bases to the Study of Quasicyclic Codes“. Ph.d.-afh. Department of Mathematics, National University of Ireland.

Lee, **Pil J. og Ernest F. Brickell** (1988). „An observation on the security of McEliece's public-key cryptosystem“. I: *Advances in cryptology—EUROCRYPT '88 (Davos, 1988)*. Bd. 330. Lecture Notes in Comput. Sci. Springer, Berlin, s. 275–280.

Leon, **Jeffrey S.** (1988). „A probabilistic algorithm for computing minimum weights of large error-correcting codes“. I: *IEEE Trans. Inform. Theory* 34.5, part 2. Coding techniques and coding theory, s. 1354–1359.

MacWilliams, **Florence Jessie og Neil James Alexander Sloane** (1977). *The theory of error-correcting codes. II*. North-Holland Mathematical Library, Vol. 16. North-Holland Publishing Co., Amsterdam-New York-Oxford, i–ix and 370–762. ISBN: 0-444-85010-4.

McEliece, **Robert James** (1978). „A Public-Key Cryptosystem Based On Algebraic Coding Theory“. I: *Deep Space Network Progress Report*, s. 114–116.

Menezes, **Alfred J.**, **Paul C. van Oorschot og Scott A. Vanstone** (1997). *Handbook of applied cryptography*. CRC Press Series on Discrete Mathematics and its Applications. With a foreword by Ronald L. Rivest. CRC Press, Boca Raton, FL, s. xxviii+780. ISBN: 0-8493-8523-7.

Minder, **Lorenz og Alistair Sinclair** (2012). „The Extended K-tree Algorithm“. I: *Journal of Cryptology* 25.2, s. 349–382.

Misoczki, **Rafael** (2013). „Two Approaches for Achieving Efficient Code-Based Cryptosystems“. Ph.d.-afh. Université Pierre et Marie Curie - Paris VI.

Márquez-Corbella, **Irene** m.fl. (2015). *Inria MOOC kursus*. URL: https://www.france-universite-numerique-mooc.fr/dashboard (sidst set 30.09.2015).

Mullen, **Gary L.**, red. (2013). *Handbook of finite fields*. Discrete Mathematics and its Applications (Boca Raton). CRC Press, Boca Raton, FL. ISBN: 978-1-4398-7378-6.

Niederreiter, **Harald** (1986). „Knapsack-type cryptosystems and algebraic coding theory“. I: *Problems Control Inform. Theory/Problemy Upravlen. Teor. Inform.* 15.2, s. 159–166.

Otmani, **Ayoub**, **Jean-Pierre Tillich og Léonard Dallot** (2010). „Cryptanalysis of two McEliece cryptosystems based on quasi-cyclic codes“. I: *Math. Comput. Sci.* 3.2, s. 129–140.

Overbeck, **Raphael og Nicolas Sendrier** (2009). „Code-based cryptography“. I: *Post-quantum cryptography*. Springer, Berlin, s. 95–145.

Özbudak, **Ferruh og Henning Stichtenoth** (2002). „Note on Niederreiter-Xing's propagation rule for linear codes“. I: *Appl. Algebra Engrg. Comm. Comput.* 13.1, s. 53–56.

Peters, **Christiane** (2009). „Explicit Bounds for Generic Decoding Algorithms for Code-Based Cryptography“. I: EIPSI Seminar, 1. april 2009.

Peters, **Christiane** (2011). „Curves, Codes, and Cryptography“. Ph.d.-afh. Technische Universiteit Eindhoven, the Netherlands.

Petrank, **Erez og Ron M. Roth** (1997). „Is code equivalence easy to decide?“ I: *IEEE Trans. Inform. Theory* 43.5, s. 1602–1604.

Prange, **Eugene** (1962). „The use of information sets in decoding cyclic codes“. I: *IRE Trans.* IT-8, S 5–S 9.

Roering, **Christopher** (2013). „Coding Theory-Based Cryptopraphy: McEliece Cryptosystems in Sage“. College of Saint Benedict/Saint John's University.

Sendrier, **Nicolas** (1998). „On the concatenated structure of a linear code“. I: *Appl. Algebra Engrg. Comm. Comput.* 9.3, s. 221–242.

Sendrier, **Nicolas** (2000). „Finding the permutation between equivalent linear codes: the support splitting algorithm“. I: *IEEE Trans. Inform. Theory* 46.4, s. 1193–1203.

Sidelnikov, **V. M. og S. O. Shestakov** (1992). „On insecurity of cryptosystems based on generalized Reed-Solomon codes“. I: *Discrete Mathematics and Applications* 2.4.

Stern, **Jacques** (1989). „A method for finding codewords of small weight“. I: *Coding theory and applications (Toulon, 1988)*. Bd. 388. Lecture Notes in Comput. Sci. Springer, New York, s. 106–113.

Wagner, **David** (2002). „A generalized birthday problem (extended abstract)“. I: *Advances in cryptology—CRYPTO 2002*. Bd. 2442. Lecture Notes in Comput. Sci. Springer, Berlin, s. 288–303.

Wieschebrink, **Christian** (2010). „Cryptanalysis of the Niederreiter public key scheme based on GRS subcodes“. I: *Post-quantum cryptography*. Bd. 6061. Lecture Notes in Comput. Sci. Springer, Berlin, s. 61–72.

APPENDIKS A

Supplerende resultater

A.1 Dekodningsalgoritme til Reed-Solomon-koder

Algoritme 16 Algoritme 5.2.1 i (Justesen m.fl., 2004).

Input: Modtaget ord $\mathbf{r} = (\ r_1\ \ldots\ r_n\ )$ samt værdierne $\ell_0 = n - t - 1$ og $\ell_1 = t$.

Output: Kodeordet $(\ f(x_1)\ \ldots\ f(x_n)\ )$ eller "Failure"

1: Løs ligningssystemet

$$\begin{bmatrix} 1 & x_1 & \ldots & x_1^{l_0} & r_1 & r_1x_1 & \ldots & r_1x_1^{l_1} \\ 1 & x_2 & \ldots & x_2^{l_0} & r_2 & r_2x_2 & \ldots & r_2x_2^{l_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \ldots & x_n^{l_0} & r_n & r_nx_n & \ldots & r_nx_n^{l_1} \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ \vdots \\ Q_{0,l_0} \\ Q_{1,0} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{A.1}$$

2: Lad $Q_0(x)$ og $Q_1(x)$ være polynomierne med koefficienterne fra ligningssystemet i punkt 1, og sæt $f(x) = -\frac{Q_0(x)}{Q_1(x)}$.

3: **if** $f(x) \in \mathbb{F}_q[x]$ **then**

4: **return** $(\ f(x_1)\ \ldots\ f(x_n)\ )$

5: **else**

6: **return** "Failure"

7: **end if**

Det problem, der kan opstå i linje 3, er, at $Q_1(x)$ ikke deler $Q_0(x)$, så $f \in \mathbb{F}_q(x) \setminus \mathbb{F}_q[x]$, hvor $\mathbb{F}_q(x)$ betegner de rationelle polynomier over $\mathbb{F}_q$. Dermed vil det ikke være muligt at rette fejlen, og det bemærkes, at dette betyder, at der er sket flere end t fejl.

A.2 Definition af MDS-koder

A.2.1 Definition (MDS-koder fra (MacWilliams m.fl., 1977) side 317):

Lineære koder, der opfylder Singleton grænsen, $d \leq n - k + 1$ ved lighed kaldes Maximum distance separable eller MDS.

En MDS-kode har derved den størst mulige afstand mellem kodeordene, og meddelelser kan systematisk indkodes til kodeord.

A.3 Generaliserede Reed-Solomon-koder

Dette afsnit bygger på (MacWilliams m.fl., 1977) kapitel 10 §8.

A.3.1 Definition:

Lad $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}$ *hvor* x_i *er forskellige elementer i* $\mathbb{F}_{q^m}$ *og lad*

$$\mathbf{v} = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$$

hvor v_i *er ikkenul elementer af* $\mathbb{F}_{q^m}$*. Den generaliserede Reed-Solomon-kode, skrives* $GRS_k(\mathbf{x}, \mathbf{v})$ *og består af alle vektorer*

$$\begin{bmatrix} v_1 f(x_1) & v_2 f(x_2) & \dots & v_n f(x_n), \end{bmatrix}$$

hvor $f(z)$ *er et polynomium med grad mindre end* k *og koefficienter fra* $\mathbb{F}_{q^m}$*. Dette er en* $[n, k, d]-$*kode over* $\mathbb{F}_{q^m}$*.*

Da f har højest $k-1$ nulpunkter er minimumsafstanden mindst $n-k+1$, derudover haves fra singletongrænsen $d \leq n-k+1$, hvorfor den er $d = n-k+1$, og dermed MDS. En generatormatrix for en generaliseret Reed-Solomon-kode er på formen

$$G = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \\ v_1 x_1 & v_2 x_2 & \cdots & v_n x_n \\ v_1 x_1^2 & v_2 x_2^2 & \cdots & v_n x_n^2 \\ \vdots & \vdots & \cdots & \vdots \\ v_1 x_1^{k-1} & v_2 x_2^{k-1} & \cdots & v_n x_n^{k-1} \end{bmatrix}.$$

A.4 Definition af alternerende koder

A.4.1 Definition (subfield subkode fra (Mullen, 2013) side 668):

Lad $C \subseteq \mathbb{F}_{q^m}^n$ *være en lineær kode. Delkoden*

$$C(sub) = \{\mathbf{c} \in C \mid \mathbf{c} \in \mathbb{F}_q^n\}$$

kaldes en subfield subkode af C*.* $C(sub)$ *er en lineær kode over* $\mathbb{F}_q$*.*

A.4.2 Definition (Alternerende koder fra (MacWilliams m.fl., 1977) side 334):

Lad $\alpha_1, \dots, \alpha_n$ *være forskellige elementer i* $\mathbb{F}_{q^m}$ *og lad* $y_1, \dots, y_n$ *være ikkenul elementer i* $\mathbb{F}_{q^m}$*. Den alternerende kode* $\mathcal{A}(\boldsymbol{\alpha}, y)$ *består af alle kodeord af* $GRS_k(\boldsymbol{\alpha}, \mathbf{v})$*, som har komponenter fra* $\mathbb{F}_q$*. Dermed er* $\mathcal{A}(\boldsymbol{\alpha}, y)$ *restriktionen af* $GRS_k(\boldsymbol{\alpha}, \mathbf{v})$ *til* $\mathbb{F}_q$*.*

$$\mathcal{A}(\boldsymbol{\alpha}, y) = \{\mathbf{a} \in GRS_k(\boldsymbol{\alpha}, \mathbf{v}) \ \mid \ \mathbf{a} \in \mathbb{F}_q^n\}$$

Den alternerende kode $\mathcal{A}(\boldsymbol{\alpha}, y)$ *består af alle vektorer,* $\mathbf{a}$ *over* $\mathbb{F}_q$*, sådan at* $H\mathbf{a}^T = 0$*, hvor* H *er paritetstjekmatricen for* $GRS_k(\boldsymbol{\alpha}, \mathbf{v})$*.*

Appendiks B
Beregninger og SAGE-kode

B.1 Kode til eksempel 1.1.1

```
#Algoritme 1
G=matrix([[1,1,1,1,1,1,1,1,1,1,1,1,1],[1,2,4,8,3,6,12,11,9,5,10,7],
[1^2,2^2,4^2,8^2,3^2,6^2,12^2,11^2,9^2,5^2,10^2,7^2]])%13
print "G="
print(G)
P=matrix([[0,0,0,1,0,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,0,1,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,1,0],
[0,0,0,0,0,1,0,0,0,0,0,0],[1,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,1,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,1],
[0,0,0,0,0,0,1,0,0,0,0,0],[0,0,1,0,0,0,0,0,0,0,0,0]])
print "P="
print(P)
S=matrix([[4,0,0],[5,11,9],[2,7,0]])
print "S="
print(S)
Ghat= S*G*P % 13
print "Ghat="
print Ghat
print ""
print "Ghat på reduceret række echelon-form"
Ghat.echelon_form()
```

Dette giver outputtet

```
G=
[ 1  1  1  1  1  1  1  1  1  1  1  1  1]
[ 1  2  4  8  3  6 12 11  9  5 10  7]
[ 1  4  3 12  9 10  1  4  3 12  9 10]
P=
[0 0 0 1 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 1 0 0 0 0 0 0]
[1 0 0 0 0 0 0 0 0 0 0 0]
```

```
[0 0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 1 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0]
S=
[ 4  0  0]
[ 5 11  9]
[ 2  7  0]
Ghat=
[ 4  4  4  4  4  4  4  4  4  4  4  4]
[ 6 11  3 12  2  3  1  6  1 11  5 12]
[ 1  3 12  9 10  8  7  6  0  4  5 11]
Ghat på reduceret række echelon-form
[ 1  0  0 11 10  5  5 11  4  2 10  2]
[ 0  1  0 10  1  3  6  6  8 10  3  2]
[ 0  0  1  6  3  6  3 10  2  2  1 10]
```

B.2 Kode til eksempel 1.1.2

```
#Algoritme 2
m=matrix([[4,7,9]])
print "m="
print m
mGhat= m*Ghat %13
print "mGhat="
print mGhat
e= matrix([[0,0,7,0,0,0,2,1,0,0,11,0]])
print "e="
print e
y=(mGhat+e) %13
print "y="
print y
```

Dette giver outputtet

```
m=
[4 7 9]
mGhat=
[ 2  3  2 12  3  5  8  8 10 12  5  4]
e=
[ 0  0  7  0  0  0  2  1  0  0 11  0]
y=
[ 2  3  9 12  3  5 10  9 10 12  3  4]
```

B.3 Kode til eksempel 1.1.3

```
#Algoritme 3
#find P invers/transponeret

Pinv= P^(-1)
print "P^(-1)="
print Pinv

```

```
yhat= y* Pinv % 13
print "yhat="
print yhat
```

Dette giver outputtet

```
P^(-1)=
[0 0 0 0 0 0 0 1 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 1]
[1 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 1 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 1 0 0]
yhat=
[12  3 12  9  3  3  5  2 10  4 10  9]
```

Ligningssystemet fra algoritmen i appendiks A.1 beregnes og løses.

```
#Dekodning af Reed-Solomon-kode

ligsys= matrix(QQ
    ,[[1,1,1^2,1^3,1^4,1^5,1^6,1^7,12,12*1,12*1^2,12*1^3,12*1^4],
[1,2,2^2,2^3,2^4,2^5,2^6,2^7,3,3*2,3*2^2,3*2^3,3*2^4],
[1,4,4^2,4^3,4^4,4^5,4^6,4^7,12,12*4,12*4^2,12*4^3,12*4^4],
[1,8,8^2,8^3,8^4,8^5,8^6,8^7,9,9*8,9*8^2,9*8^3,9*8^4],
[1,3,3^2,3^3,3^4,3^5,3^6,3^7,3,3*3,3*3^2,3*3^3,3*3^4],
[1,6,6^2,6^3,6^4,6^5,6^6,6^7,3,3*6,3*6^2,3*6^3,3*6^4],
[1,12,12^2,12^3,12^4,12^5,12^6,12^7,5,5*12,5*12^2,5*12^3,5*12^4],
[1,11,11^2,11^3,11^4,11^5,11^6,11^7,2,2*11,2*11^2,2*11^3,2*11^4],
[1,9,9^2,9^3,9^4,9^5,9^6,9^7,10,10*9,10*9^2,10*9^3,10*9^4],
[1,5,5^2,5^3,5^4,5^5,5^6,5^7,4,4*5,4*5^2,4*5^3,4*5^4],
[1,10,10^2,10^3,10^4,10^5,10^6,10^7,10,10*10,10*10^2,10*10^3,10*10^4],
[1,7,7^2,7^3,7^4,7^5,7^6,7^7,9,9*7,9*7^2,9*7^3,9*7^4]])%13
print "ligsys="
print ligsys

rre= ligsys.echelon_form()%13
print "ligningssystem på reduceret-række-echelon-form:"
print rre
```

Dette giver outputtet

```
ligsys=
[ 1  1  1  1  1  1  1  1 12 12 12 12 12]
[ 1  2  4  8  3  6 12 11  3  6 12 11  9]
[ 1  4  3 12  9 10  1  4 12  9 10  1  4]
[ 1  8 12  5  1  8 12  5  9  7  4  6  9]
[ 1  3  9  1  3  9  1  3  3  9  1  3  9]
[ 1  6 10  8  9  2 12  7  3  5  4 11  1]
```

```
[ 1 12  1 12  1 12  1 12  5  8  5  8  5]
[ 1 11  4  5  3  7 12  2  2  9  8 10  6]
[ 1  9  3  1  9  3  1  9 10 12  4 10 12]
[ 1  5 12  8  1  5 12  8  4  7  9  6  4]
[ 1 10  9 12  3  4  1 10 10  9 12  3  4]
[ 1  7 10  5  9 11 12  6  9 11 12  6  3]

ligningssystem på reduceret-række-echelon-form:

[ 1  0  0  0  0  0  0  0  0  0  0  0 11]
[ 0  1  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  1  0  0  0  0  0  0  0  0  0  1]
[ 0  0  0  1  0  0  0  0  0  0  0  0  8]
[ 0  0  0  0  1  0  0  0  0  0  0  0  9]
[ 0  0  0  0  0  1  0  0  0  0  0  0  7]
[ 0  0  0  0  0  0  1  0  0  0  0  0 11]
[ 0  0  0  0  0  0  0  1  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  1  0  0  0  7]
[ 0  0  0  0  0  0  0  0  0  1  0  0  2]
[ 0  0  0  0  0  0  0  0  0  0  1  0  8]
[ 0  0  0  0  0  0  0  0  0  0  0  1  5]
```

Polynomierne $Q_0(x)$ og $Q_1(x)$ dannes.

```
x = PolynomialRing(GF(13),'x').gen()
Q0=2+12*x^2+5*x^3+4*x^4+6*x^5+2*x^6
print "Q_0="
print Q0
Q1=6+11*x+5*x^2+8*x^3+x^4
print "Q_1="
print Q1

minusf= Q0.quo_rem(Q1)
print "-f="
print minusf[0]

f= -(minusf[0])
print "f="
print f

Sinv= S^(-1) %13
print "S^(-1)="
print Sinv
msendt= matrix([[4,10,11]])*Sinv % 13
print "meddelelse="
print msendt
```

Dette giver outputtet

```
Q_0=
2*x^6 + 6*x^5 + 4*x^4 + 5*x^3 + 12*x^2 + 2
Q_1=
x^4 + 8*x^3 + 5*x^2 + 11*x + 6
-f=
```

```
2*x^2 + 3*x + 9
f=
11*x^2 + 10*x + 4
S^(-1)=
[10  0  0]
[12  0  2]
[ 0  3 12]
meddelelse=
[4 7 9]
```

B.4 Kode til eksempel 1.2.3

```
H=matrix(GF(2),[[1,1,0,0,0,0,0,0],[0,0,0,1,0,1,1,1],[0,0,1,1,1,0,0,1],
[0,1,1,1,1,1,1,1],[0,0,1,0,1,1,0,1],[0,0,0,1,1,1,1,0]])
print"H ="
print H
print"Basis findes:"
H.right_kernel()
```

Dette giver outputtet

```
H =
[1 1 0 0 0 0 0 0]
[0 0 0 1 0 1 1 1]
[0 0 1 1 1 0 0 1]
[0 1 1 1 1 1 1 1]
[0 0 1 0 1 1 0 1]
[0 0 0 1 1 1 1 0]
Basis findes:
Vector space of degree 8 and dimension 2 over Finite Field of size 2
Basis matrix:
[1 1 0 0 1 0 1 1]
[0 0 1 1 1 1 1 1]
```

B.5 Kode til eksempel 1.3.1

```
#Algoritme Nieterreiter keys
H=matrix([[1,2,4,8,3,6,12,11,9,5,10,7],
[1^2,2^2,4^2,8^2,3^2,6^2,12^2,11^2,9^2,5^2,10^2,7^2],
[1^3,2^3,4^3,8^3,3^3,6^3,12^3,11^3,9^3,5^3,10^3,7^3],
[1^4,2^4,4^4,8^4,3^4,6^4,12^4,11^4,9^4,5^4,10^4,7^4],
[1^5,2^5,4^5,8^5,3^5,6^5,12^5,11^5,9^5,5^5,10^5,7^5],
[1^6,2^6,4^6,8^6,3^6,6^6,12^6,11^6,9^6,5^6,10^6,7^6],
[1^7,2^7,4^7,8^7,3^7,6^7,12^7,11^7,9^7,5^7,10^7,7^7],
[1^8,2^8,4^8,8^8,3^8,6^8,12^8,11^8,9^8,5^8,10^8,7^8],
[1^9,2^9,4^9,8^9,3^9,6^9,12^9,11^9,9^9,5^9,10^9,7^9]])%13
print "H="
print H
P=matrix([[0,0,0,1,0,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,0,1,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,1,0],
[0,0,0,0,0,1,0,0,0,0,0,0],[1,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,1,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,1],
```

```
[0,0,0,0,0,0,1,0,0,0,0,0],[0,0,1,0,0,0,0,0,0,0,0,0]])
print "P="
print(P)
S=matrix([[4,0,11,11,0,2,3,0,0],[5,11,0,11,0,5,9,0,3],
[2,7,4,0,10,0,4,0,5],[5,12,8,4,0,0,5,1,0],[10,11,4,2,7,0,0,2,0],
[1,3,11,0,6,9,0,4,0],[4,2,5,11,3,7,7,0,3],[2,8,4,9,1,4,5,2,0],
[5,5,2,12,6,8,4,0,7]])
print "S="
print S
Hhat= S*H*P % 13
print "Hhat="
print Hhat
```

Dette giver outputtet

```
H=
[ 1  2  4  8  3  6 12 11  9  5 10  7]
[ 1  4  3 12  9 10  1  4  3 12  9 10]
[ 1  8 12  5  1  8 12  5  1  8 12  5]
[ 1  3  9  1  3  9  1  3  9  1  3  9]
[ 1  6 10  8  9  2 12  7  3  5  4 11]
[ 1 12  1 12  1 12  1 12  1 12  1 12]
[ 1 11  4  5  3  7 12  2  9  8 10  6]
[ 1  9  3  1  9  3  1  9  3  1  9  3]
[ 1  5 12  8  1  5 12  8  1  5 12  8]
P=
[0 0 0 1 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 1 0 0 0 0 0 0]
[1 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 1 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0]
S=
[ 4  0 11 11  0  2  3  0  0]
[ 5 11  0 11  0  5  9  0  3]
[ 2  7  4  0 10  0  4  0  5]
[ 5 12  8  4  0  0  5  1  0]
[10 11  4  2  7  0  0  2  0]
[ 1  3 11  0  6  9  0  4  0]
[ 4  2  5 11  3  7  7  0  3]
[ 2  8  4  9  1  4  5  2  0]
[ 5  5  2 12  6  8  4  0  7]

Hhat=
[ 6  4  3  5  2  8  3  7  6  1  9 11]
[ 0  1  5  5  0 10  1  0  6  8  0  3]
[ 6 11  5  6 11  8 11  0 10  6  5 12]
```

```
[ 5  3  4  9 11 12  0  7  4  3  2  5]
[ 0  6  2 10 12  7  0  6  3  7  6  6]
[ 4  9  5  8 10 11  4 12  3  5  9 11]
[ 6 10  4  3  9 11  3 12  7  9 12  5]
[ 2  1  4  9 12 11  9  3  3  6 10  8]
[11  7  3 10 10  1 12  4  5 10 11  7]
```

B.6 Kode til eksempel 1.3.2

```
# Algoritme Niederreiter encoding

x=matrix([[0,0,0,4,0,0,7,0,0,0,0,9]])
print "x="
print x
xT=x.transpose()
s= Hhat*xT %13
print "s="
print s
```

```
x=
[0 0 0 4 0 0 7 0 0 0 0 9]
s=
[10]
[ 2]
[ 1]
[ 3]
[ 3]
[ 3]
[ 0]
[ 2]
[ 5]
```

B.7 Kode til eksempel 1.3.3

```
#Algoritme Niederreiter dekodning

Ss=S^(-1)*s%13
print "S^(-1)*s="
print Ss


zT= H.solve_right(Ss) %13
print "z^T="
print zT
z=transpose(zT)
print "z="
print z
```

Dette giver outputtet

```
S^(-1)*s=
[ 2]
[ 6]
```

```
[ 4]
[ 8]
[12]
[ 2]
[ 3]
[11]
[ 3]
z^T=
[ 4]
[ 9]
[11]
[ 3]
[ 4]
[ 9]
[ 5]
[ 6]
[ 5]
[ 0]
[ 0]
[ 0]
z=
[ 4  9 11  3  4  9  5  6  5  0  0  0]
```

Ligningssystemet fra algoritmen i appendiks A.1 beregnes og løses.

```
#Dekodning af Reed-Solomon-kode

ligsys= matrix(QQ,[[1,1,1^2,1^3,1^4,1^5,1^6,1^7,4,4*1,4*1^2,4*1^3,4*1^4],
[1,2,2^2,2^3,2^4,2^5,2^6,2^7,9,9*2,9*2^2,9*2^3,9*2^4],
[1,4,4^2,4^3,4^4,4^5,4^6,4^7,11,11*4,11*4^2,11*4^3,11*4^4],
[1,8,8^2,8^3,8^4,8^5,8^6,8^7,3,3*8,3*8^2,3*8^3,3*8^4],
[1,3,3^2,3^3,3^4,3^5,3^6,3^7,4,4*3,4*3^2,4*3^3,4*3^4],
[1,6,6^2,6^3,6^4,6^5,6^6,6^7,9,9*6,9*6^2,9*6^3,9*6^4],
[1,12,12^2,12^3,12^4,12^5,12^6,12^7,5,5*12,5*12^2,5*12^3,5*12^4],
[1,11,11^2,11^3,11^4,11^5,11^6,11^7,6,6*11,6*11^2,6*11^3,6*11^4],
[1,9,9^2,9^3,9^4,9^5,9^6,9^7,5,5*9,5*9^2,5*9^3,5*9^4],
[1,5,5^2,5^3,5^4,5^5,5^6,5^7,0,0*5,0*5^2,0*5^3,0*5^4],
[1,10,10^2,10^3,10^4,10^5,10^6,10^7,0,0*10,0*10^2,0*10^3,0*10^4],
[1,7,7^2,7^3,7^4,7^5,7^6,7^7,0,0*7,0*7^2,0*7^3,0*7^4]])%13
print "ligsys="
print ligsys

rre= ligsys.echelon_form()%13
print "ligningssystem på reduceret-række-echelon-form:"
print rre
```

Dette giver outputtet

```
ligsys=
[ 1  1  1  1  1  1  1  1  4  4  4  4  4]
[ 1  2  4  8  3  6 12 11  9  5 10  7  1]
[ 1  4  3 12  9 10  1  4 11  5  7  2  8]
[ 1  8 12  5  1  8 12  5  3 11 10  2  3]
[ 1  3  9  1  3  9  1  3  4 12 10  4 12]
```

```
[ 1  6 10  8  9  2 12  7  9  2 12  7  3]
[ 1 12  1 12  1 12  1 12  5  8  5  8  5]
[ 1 11  4  5  3  7 12  2  6  1 11  4  5]
[ 1  9  3  1  9  3  1  9  5  6  2  5  6]
[ 1  5 12  8  1  5 12  8  0  0  0  0  0]
[ 1 10  9 12  3  4  1 10  0  0  0  0  0]
[ 1  7 10  5  9 11 12  6  0  0  0  0  0]
ligningssystem på reduceret-række-echelon-form:
[ 1  0  0  0  0  0  0  0  0  0  0  0  5]
[ 0  1  0  0  0  0  0  0  0  0  0  0  4]
[ 0  0  1  0  0  0  0  0  0  0  0  0  4]
[ 0  0  0  1  0  0  0  0  0  0  0  0  2]
[ 0  0  0  0  1  0  0  0  0  0  0  0  1]
[ 0  0  0  0  0  1  0  0  0  0  0  0  4]
[ 0  0  0  0  0  0  1  0  0  0  0  0  6]
[ 0  0  0  0  0  0  0  1  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  1  0  0  0  7]
[ 0  0  0  0  0  0  0  0  0  1  0  0 11]
[ 0  0  0  0  0  0  0  0  0  0  1  0  9]
[ 0  0  0  0  0  0  0  0  0  0  0  1  0]
```

Polynomierne $Q_0(x)$ og $Q_1(x)$ dannes.

```
x = PolynomialRing(GF(13),'x').gen()
Q0=8+9*x+9*x^2+11*x^3+12*x^4+9*x^5+7*x^6
print "Q_0="
print Q0
Q1=6+2*x+4*x^2+x^4
print "Q_1="
print Q1

minusf= Q0.quo_rem(Q1)
print "-f="
print minusf[0]

f= -(minusf[0])
print "f="
print f
```

Dette giver outputtet

```
Q_0=
7*x^6 + 9*x^5 + 12*x^4 + 11*x^3 + 9*x^2 + 9*x + 8
Q_1=
x^4 + 4*x^2 + 2*x + 6
-f=
7*x^2 + 9*x + 10
f=
6*x^2 + 4*x + 3
```

Kodeordet c beregnes ved af indsætte x_i'erne i $f(x)$.

```
c=matrix([[0,9,11,3,4,9,5,6,5,4,6,0]])
print "c="
print c
```

```
4
5  xsendt=(z-c)*P % 13 # Vi har: c=z-xP^T --> cP=zP-x --> x=(z-c)P
6
7  print "x="
8  print xsendt
```

Dette giver outputtet

```
c=
[ 0  9 11  3  4  9  5  6  5  4  6  0]
x=
[0 0 0 4 0 0 7 0 0 0 0 9]
```

B.8 Kode til eksempel 2.1.2

Først skal en McEliece offentlig nøgle genereres. Der vælges, at der arbejdes med en Generaliseret Reed-Solomon-kode med parametrene $q = 7$, $n = 6$, $k = 2$. Dermed bliver $d = 6 - 2 + 1 = 5$, hvormed t bliver 2. En vektor $\mathbf{v}$ af ikkenul elementer i $\mathbb{F}_{q^m}$ vælges som følgende

$$\mathbf{v} = \begin{bmatrix} 2 & 3 & 4 & 2 & 6 & 1 \end{bmatrix},$$

og $\mathbf{x}$ dannes ud fra $x_i = 3^{i-1}$.

```
1  x=[]
2  for i in range(1,7):
3      x.append(3^(i-1)%7)
4  print "x=", x
```

Dette giver outputtet

```
x= [1, 3, 2, 6, 4, 5]
```

Nu kan en generatormatrix for koden, samt den offentlige nøgle til McEliece kryptosystemet beregnes ved hjælp af algoritme 1. Dette gøres i følgende SAGE kode.

```
1  G=matrix([[2,3,4,2,6,1],[2*1,3*3,4*2,2*6,6*4,1*5]])%7
2  print "G="
3  print G
4  P=matrix([[0,0,1,0,0,0],[1,0,0,0,0,0],[0,0,0,0,1,0],[0,0,0,0,0,1],
5  [0,1,0,0,0,0],[0,0,0,1,0,0]])
6  S= matrix([[1,2],[0,1]])
7  Ghat= S*G*P % 7
8  print "Ghat="
9  print Ghat
```

Dette giver outputtet

```
G=
[2 3 4 2 6 1]
[2 2 1 5 3 5]
Ghat=
[0 5 6 4 6 5]
[2 3 2 5 1 5]
```

Nu benyttes algoritme 2 til at danne cifferteksten, som ønskes brudt ved meddelelsesangreb.

```
#Algoritme 2
m=matrix([[4,0]])

mGhat= m*Ghat %7
print "mGhat=", mGhat
e= matrix([[0,4,0,0,2,0]])

y=(mGhat+e) %7
print "y=", y
```

Dette giver outputtet

```
mGhat= [0 6 3 2 3 6]
y= [0 3 3 2 5 6]
```

Et information set vælges, G', $y_{\mathcal{I}}$ samt et nyt y dannes.

```
GhatI=matrix([[0,5],[2,3]])%7
GhatIinv= GhatI^(-1)%7
Gmerke= GhatIinv*Ghat % 7
print "G'="
print Gmerke
yI=matrix([[0,3]])
print "y_I=", yI
nyy= (y-yI*Gmerke) %7
print "y=", nyy
```

Dette giver outputtet

```
G'=
[1 0 2 2 5 1]
[0 1 4 5 4 1]
y_I= [0 3]
y= [0 0 5 1 0 3]
```

Der dannes nu en ny G', hvor der vælges et nyt information set, da algoritmen med det andet information set ikke gav en fejlvektor med vægt mindre end lig t.

```
GhatI=matrix([[5,6],[3,1]])%7
GhatIinv= GhatI^(-1)%7
Gmerke= GhatIinv*Ghat % 7
print "G'="
print Gmerke
yI=matrix([[3,5]])
print "y_I=", yI
nyy= (y-yI*Gmerke) %7
print "y=", nyy
```

```
G'=
[2 1 1 2 0 3]
[3 0 6 6 1 3]
y_I= [3 5]
y= [0 0 5 1 0 3]
```

Beregningerne af punkt 8 i algoritme 7 med information set $\mathcal{I} = \{2,5\}$ ses i nedenstående tabel 8.

m_1	$\mathbf{r}_1$	$\phi_{\mathbf{m}}(A) = \mathbf{y} - \mathbf{r_1}$
1	$\begin{bmatrix}1\end{bmatrix}\begin{bmatrix}2&1&1&2&0&3\end{bmatrix} = \begin{bmatrix}2&1&1&2&0&3\end{bmatrix}$	$\begin{bmatrix}0&5\end{bmatrix} - \begin{bmatrix}2&1\end{bmatrix} = \begin{bmatrix}5&4\end{bmatrix}$
2	$\begin{bmatrix}2\end{bmatrix}\begin{bmatrix}2&1&1&2&0&3\end{bmatrix} = \begin{bmatrix}4&2&2&4&0&6\end{bmatrix}$	$\begin{bmatrix}0&5\end{bmatrix} - \begin{bmatrix}4&2\end{bmatrix} = \begin{bmatrix}3&3\end{bmatrix}$
3	$\begin{bmatrix}3\end{bmatrix}\begin{bmatrix}2&1&1&2&0&3\end{bmatrix} = \begin{bmatrix}6&3&3&6&0&2\end{bmatrix}$	$\begin{bmatrix}0&5\end{bmatrix} - \begin{bmatrix}6&3\end{bmatrix} = \begin{bmatrix}1&2\end{bmatrix}$
4	$\begin{bmatrix}4\end{bmatrix}\begin{bmatrix}2&1&1&2&0&3\end{bmatrix} = \begin{bmatrix}1&4&4&1&0&5\end{bmatrix}$	$\begin{bmatrix}0&5\end{bmatrix} - \begin{bmatrix}1&4\end{bmatrix} = \begin{bmatrix}6&1\end{bmatrix}$
5	$\begin{bmatrix}5\end{bmatrix}\begin{bmatrix}2&1&1&2&0&3\end{bmatrix} = \begin{bmatrix}3&5&5&3&0&1\end{bmatrix}$	$\begin{bmatrix}0&5\end{bmatrix} - \begin{bmatrix}3&5\end{bmatrix} = \begin{bmatrix}4&0\end{bmatrix}$
6	$\begin{bmatrix}6\end{bmatrix}\begin{bmatrix}2&1&1&2&0&3\end{bmatrix} = \begin{bmatrix}5&6&6&5&0&4\end{bmatrix}$	$\begin{bmatrix}0&5\end{bmatrix} - \begin{bmatrix}5&6\end{bmatrix} = \begin{bmatrix}2&6\end{bmatrix}$

Tabel 8. Udregningerne fra punkt 8 i algoritme 7

Beregningerne af punkt 9 i algoritme 7 med information set $\mathcal{I} = \{2,5\}$ ses i nedenstående tabel 9.

m_1'	$\mathbf{r}_1$	$\psi_{\mathbf{m}'}(B) = \mathbf{r_1}'$
1	$\begin{bmatrix}1\end{bmatrix}\begin{bmatrix}3&0&6&6&1&3\end{bmatrix} = \begin{bmatrix}3&0&6&6&1&3\end{bmatrix}$	$\begin{bmatrix}3&6\end{bmatrix}$
2	$\begin{bmatrix}2\end{bmatrix}\begin{bmatrix}3&0&6&6&1&3\end{bmatrix} = \begin{bmatrix}6&0&5&5&2&6\end{bmatrix}$	$\begin{bmatrix}6&5\end{bmatrix}$
3	$\begin{bmatrix}3\end{bmatrix}\begin{bmatrix}3&0&6&6&1&3\end{bmatrix} = \begin{bmatrix}2&0&4&4&3&2\end{bmatrix}$	$\begin{bmatrix}2&4\end{bmatrix}$
4	$\begin{bmatrix}4\end{bmatrix}\begin{bmatrix}3&0&6&6&1&3\end{bmatrix} = \begin{bmatrix}5&0&3&3&4&5\end{bmatrix}$	$\begin{bmatrix}5&3\end{bmatrix}$
5	$\begin{bmatrix}5\end{bmatrix}\begin{bmatrix}3&0&6&6&1&3\end{bmatrix} = \begin{bmatrix}1&0&2&2&5&1\end{bmatrix}$	$\begin{bmatrix}1&2\end{bmatrix}$
6	$\begin{bmatrix}6\end{bmatrix}\begin{bmatrix}3&0&6&6&1&3\end{bmatrix} = \begin{bmatrix}4&0&1&1&6&4\end{bmatrix}$	$\begin{bmatrix}4&1\end{bmatrix}$

Tabel 9. Udregningerne fra punkt 9 i algoritme 7

Det undersøges, om kodeordet ligger i koden.

```
Ghat=matrix(QQ,[[0,5,6,4,6,5],[2,3,2,5,1,5]])
print "Ghat på reduceret række-echelon-form"
Ghat.echelon_form()%7
H=matrix([[-2,-4,1,0,0,0],[-2,-5,0,1,0,0],[-5,-4,0,0,1,0],[-1,-1,0,0,0,1]])
    %7
print "H="
print H
c=matrix([[0,6,3,2,3,6]])%7
Hc= H*transpose(c)%7
print "Hc^T="
print Hc
```

Dette giver outputtet

```
Ghat på reduceret række-echelon-form
[1 0 2 2 5 1]
[0 1 4 5 4 1]
```

```
H=
[5 3 1 0 0 0]
[5 2 0 1 0 0]
[2 3 0 0 1 0]
[6 6 0 0 0 1]
Hc^T=
[0]
[0]
[0]
[0]
```

Nu findes meddelelsen ved brug af ligning (2.1).

```
ci=matrix([[6,3]])
print "c_I=" , ci
m=ci*GhatIinv%7
print "m=" , m
```

Dette giver outputtet

```
c_I= [6 3]
m= [4 0]
```

B.9 Kode til generering af $\hat{G}$ benyttet ved nøgleangreb

Generatormatricen fra eksempel 1.2.4 benyttes.

```
G=matrix(GF(2),[[0,0,1,1,1,1,1,1],[1,1,0,0,1,0,1,1]])
S=matrix(GF(2),[[1,0],[0,1]])
P=matrix(GF(2),[[0,0,0,0,0,1,0,0],[0,1,0,0,0,0,0,0],[0,0,0,1,0,0,0,0],
[1,0,0,0,0,0,0,0],[0,0,0,0,0,0,1,0],[0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,1],
[0,0,0,0,1,0,0,0]])
Ghat=S*G*P
print "Ghat ="
print Ghat
```

Dette giver outputtet

```
Ghat =
[1 0 1 1 1 0 1 1]
[0 1 0 0 1 1 1 1]
```

B.10 Kode til eksempel 3.2.7

```
MS = MatrixSpace(GF(2),4,14)
G = MS([[1,0,1,1,1,0,1,0,0,1,1,1,1,0],[0,1,1,1,0,0,0,0,0,0,1,0,1,0],
[0,0,1,1,1,1,0,1,0,1,1,1,0,1],[0,0,0,1,0,1,0,0,1,1,1,0,0,0]])
C = LinearCode(G)
print "C =" , C.list() # Kodeordene i C printes
```

Dette giver følgende output, der er de 16 kodeord i koden.

```
C = [(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0),
```

```
(0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0),
(1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0),
(0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1),
(1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1),
(0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1),
(1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1),
(0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0),
(1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0),
(0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0),
(1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0),
(0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1),
(1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1),
(0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1),
(1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1)]
```

B.11 Kode til eksempel 3.4.1

```
MS = MatrixSpace(GF(3),3,10)
G = MS([[1,1,1,1,1,0,0,0,0,0], [0,0,0,0,0,2,2,2,2,2],
    [0,0,0,2,1,0,0,0,2,1]])
C = LinearCode(G)
print "minimumsafstand =" , C.minimum_distance() #minimumsafstanden
    beregnes
print ""
print "C =" , C.list() # Kodeordene i C printes
#permutationsmatrien dannes
Pi=matrix(GF(3)
    ,[[0,1,0,0,0,0,0,0,0,0],[0,0,1,0,0,0,0,0,0,0],[0,0,0,1,0,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0],[1,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,1,0,0,0],
[0,0,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,0,0,1,0],[0,0,0,0,0,0,0,0,0,1],
[0,0,0,0,0,1,0,0,0,0]])
print "Pi="
print Pi
G2=matrix(GF(3)
    ,[[1,1,1,1,1,0,0,0,0,0],[0,0,0,2,1,0,0,0,2,1],[0,0,0,0,0,1,1,1,1,1],
[0,0,0,2,1,0,0,0,2,1]])
print "G2="
print G2
Gmerke= G2*Pi
print"G'="
print Gmerke
```

Dette giver outputtet

```
minimumsafstand = 4

C = [(0, 0, 0, 0, 0, 0, 0, 0, 0, 0), (1, 1, 1, 1, 1, 0, 0, 0, 0, 0), (2, 2,
     2, 2, 2, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 2, 2, 2, 2, 2), (1, 1, 1, 1,
    1, 2, 2, 2, 2, 2), (2, 2, 2, 2, 2, 2, 2, 2, 2, 2), (0, 0, 0, 0, 0, 1,
    1, 1, 1, 1), (1, 1, 1, 1, 1, 1, 1, 1, 1, 1), (2, 2, 2, 2, 2, 1, 1, 1,
    1, 1), (0, 0, 0, 2, 1, 0, 0, 0, 2, 1), (1, 1, 1, 0, 2, 0, 0, 0, 2, 1),
    (2, 2, 2, 1, 0, 0, 0, 0, 2, 1), (0, 0, 0, 2, 1, 2, 2, 2, 1, 0), (1, 1,
    1, 0, 2, 2, 2, 2, 1, 0), (2, 2, 2, 1, 0, 2, 2, 2, 1, 0), (0, 0, 0, 2,
```

```
    1, 1, 1, 1, 0, 2), (1, 1, 1, 0, 2, 1, 1, 1, 0, 2), (2, 2, 2, 1, 0, 1,
    1, 1, 0, 2), (0, 0, 0, 1, 2, 0, 0, 0, 1, 2), (1, 1, 1, 2, 0, 0, 0, 0,
    1, 2), (2, 2, 2, 0, 1, 0, 0, 0, 1, 2), (0, 0, 0, 1, 2, 2, 2, 2, 0, 1),
    (1, 1, 1, 2, 0, 2, 2, 2, 0, 1), (2, 2, 2, 0, 1, 2, 2, 2, 0, 1), (0, 0,
    0, 1, 2, 1, 1, 1, 2, 0), (1, 1, 1, 2, 0, 1, 1, 1, 2, 0), (2, 2, 2, 0,
    1, 1, 1, 1, 2, 0)]

Pi=
[0 1 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[1 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 1 0 0 0 0]
G2=
[1 1 1 1 1 0 0 0 0 0]
[0 0 0 2 1 0 0 0 2 1]
[0 0 0 0 0 1 1 1 1 1]
[0 0 0 2 1 0 0 0 2 1]
G'=
[1 1 1 1 1 0 0 0 0 0]
[1 0 0 0 2 1 0 0 0 2]
[0 0 0 0 0 1 1 1 1 1]
[1 0 0 0 2 1 0 0 0 2]
```

B.12 Kode til eksempel 3.4.2

```
1  M=matrix(GF(3)
       ,[[1,1,1,1,1,0,0,0,0,0],[0,0,0,0,0,1,1,1,1,1],[1,0,0,0,2,1,0,0,0,2]])
2  print "M="
3  print M
4  m=matrix(GF(3),[[2,0,1]])
5  print "m="
6  print m
7  c=m*M
8  print "c="
9  print c
10 e=matrix(GF(3),[[0,0,0,0,0,0,0,0,0,2]])
11 print "e="
12 print e
13 y=c+e
14 print "y="
15 print y
```

Dette giver outputtet

```
M=
[1 1 1 1 1 0 0 0 0 0]
[0 0 0 0 0 1 1 1 1 1]
```

```
[1 0 0 0 2 1 0 0 0 2]
m=
[2 0 1]
c=
[0 2 2 2 1 1 0 0 0 2]
e=
[0 0 0 0 0 0 0 0 0 2]
y=
[0 2 2 2 1 1 0 0 0 1]
```

B.13 Kode til eksempel 3.4.3

```
Piinv=Pi^(-1)
print "Pi^(-1)"
print Piinv
print ""
ymerke= y*Piinv
print "y'=", ymerke
cc=matrix(GF(3),[[2,2,2,1,0,0,0,0,2,1]]) #kodeord i C, der kun har afstand
    t=1 til y'
print "c=", cc
MM=M*Piinv
print "MPi^(-1)"
print MM
m=MM.solve_left(cc)
print "m=", m
```

Dette giver outputtet

```
Pi^(-1)
[0 0 0 0 1 0 0 0 0 0]
[1 0 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 1 0]

y'= [2 2 2 1 0 0 0 0 1 1]
c= [2 2 2 1 0 0 0 0 2 1]
MPi^(-1)
[1 1 1 1 1 0 0 0 0 0]
[0 0 0 0 0 1 1 1 1 1]
[0 0 0 2 1 0 0 0 2 1]
m= [2 0 1]
```

B.14 Kode til eksempel 3.3.2

```
R = singular.ring(3,'(x)','(c,lp)')
M = singular.module('[x^4+1,x^3+x^2+1,2*x^4,x^3+2*x]','[x^2+2*x,x^4+x^3+x+1,x^2+2*x,x^3+x^2+2]','[x^2+x+1,2*x^3+x,x^3+2*x^2+x+2,x^4+x]','[x^5-1,0,0,0]','[0,x^5-1,0,0]','[0,0,x^5-1,0]','[0,0,0,x^5-1]')
singular.option('redSB')
GB=singular.std(M)
print "GB ="
print GB
print ""
print "x^5-1 =" , factor(x^5-1)
print ""
s=5
r=4
g1=GB[1,4]
print "g_1^1 =" , g1
print ""
g2=GB[2,3]
print "g_2^2 =" , g2
print ""
g3=GB[3,2]
print "g_3^3 =" , g3
print ""
g4=GB[4,1]
print "g_4^4 =" , g4
print ""
k=s*r-((1+0*x).degree(x)+(1+0*x).degree(x)+(x-1).degree(x)+(x^4+x^3+x^2+x+1).degree(x))
print "k =" , k
```

Dette giver outputtet

```
GB =
0,              0,  0,      1,
0,              0,  1,      0,
0,              x-1,0,      1,
x^4+x^3+x^2+x+1,x-1,-x^3+x+1,-x^2+1

x^5-1 = (x^4 + x^3 + x^2 + x + 1)*(x - 1)

g_1^1 = 1

g_2^2 = 1

g_3^3 = x-1

g_4^4 = x^4+x^3+x^2+x+1

k = 15
```

B.15 Beregninger til eksempel 4.1.3

$$c3 = \begin{bmatrix}0&1&0\\0&0&0\\0&1&0\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}0&1&1\\0&0&0\\0&1&1\end{bmatrix}$$

$$c4 = \begin{bmatrix}0&1&1\\0&0&1\\0&1&1\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\0&1&0\end{bmatrix}$$

$$c5 = \begin{bmatrix}0&1&0\\0&1&0\\0&0&0\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}0&1&1\\0&1&1\\0&0&0\end{bmatrix}$$

$$c6 = \begin{bmatrix}0&1&1\\0&1&1\\0&0&1\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}0&1&0\\0&1&0\\0&0&1\end{bmatrix}$$

$$c7 = \begin{bmatrix}0&0&0\\0&1&0\\0&1&0\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}0&0&0\\0&1&1\\0&1&1\end{bmatrix}$$

$$c8 = \begin{bmatrix}0&0&1\\0&1&1\\0&1&1\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}0&0&1\\0&1&0\\0&1&0\end{bmatrix}$$

$$c9 = \begin{bmatrix}1&0&0\\1&0&0\\1&0&0\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$$c10 = \begin{bmatrix}1&0&1\\1&0&1\\1&0&1\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}1&1&0\\1&1&0\\1&1&0\end{bmatrix}$$

$$c11 = \begin{bmatrix}1&1&0\\1&0&0\\1&1&0\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}1&0&0\\1&1&1\\1&0&0\end{bmatrix}$$

$$c12 = \begin{bmatrix}1&1&1\\1&0&1\\1&1&1\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}1&0&1\\1&1&0\\1&0&1\end{bmatrix}$$

$$c13 = \begin{bmatrix}1&1&0\\1&1&0\\1&0&0\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}1&0&0\\1&0&0\\1&1&1\end{bmatrix}$$

$$c14 = \begin{bmatrix}1&1&1\\1&1&1\\1&0&1\end{bmatrix}\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}1&0&1\\1&0&1\\1&1&0\end{bmatrix}$$

$$c15 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$c16 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

B.16 Kode til eksempel 4.1.8

```
#Minimumsafstanden for C1 beregnes
MS=MatrixSpace(GF(3),3,4)
G1=MS([[1,1,1,1],[2,1,0,1],[0,1,1,0]])
print "G1=", G1
C=LinearCode(G1)
print C
d1=C.minimum_distance()
print "d1=", d1
```

Dette giver outputtet

```
G1=
[1 1 1 1]
[2 1 0 1]
[0 1 1 0]
Linear code of length 4, dimension 3 over Finite Field of size 3
d1= 2
```

```
#Minimumsafstanden for C2=C3 beregnes
MS=MatrixSpace(GF(3),2,4)
G2=MS([[1,1,1,1],[2,1,0,1]])
print "G2=", G2
C=LinearCode(G2)
print C
d2=C.minimum_distance()
print "d2=", d2
```

Dette giver outputtet

```
G2=
[1 1 1 1]
[2 1 0 1]
Linear code of length 4, dimension 2 over Finite Field of size 3
d2= 2
```

```
#Minimumsafstanden for C4 beregnes
MS=MatrixSpace(GF(3),1,4)
G4=MS([[1,1,1,1]])
print "G4=", G4
C=LinearCode(G4)
print C
d4=C.minimum_distance()
print "d4=", d4
```

Dette giver outputtet

```
G4= [1 1 1 1]
Linear code of length 4, dimension 1 over Finite Field of size 3
d4= 4
```

```
#Matrix A defineres
MS=MatrixSpace(GF(3),4,4)
A=MS([[1,2,1,0],[0,2,0,0],[0,0,2,2],[0,0,0,1]])
print "A="
print A
#minimumsafstanden for CR1 beregnes
MS=MatrixSpace(GF(3),1,4)
GCR1=MS([[1,2,1,0]])
print "GCR1=", GCR1
CR1=LinearCode(GCR1)
print CR1
D1=CR1.minimum_distance()
print "D1=", D1

#minimumsafstanden for CR2 beregnes
MS=MatrixSpace(GF(3),2,4)
GCR2=MS([[1,2,1,0],[0,2,0,0]])
print "GCR2=", GCR2
CR2=LinearCode(GCR2)
print CR2
D2=CR2.minimum_distance()
print "D2=", D2

#minimumsafstanden for CR3 beregnes
MS=MatrixSpace(GF(3),3,4)
GCR3=MS([[1,2,1,0],[0,2,0,0],[0,0,2,2]])
print "GCR3=", GCR3
CR3=LinearCode(GCR3)
print CR3
D3=CR3.minimum_distance()
print "D3=", D3

#minimumsafstanden for CR4 beregnes
MS=MatrixSpace(GF(3),4,4)
GCR4=MS([[1,2,1,0],[0,2,0,0],[0,0,2,2],[0,0,0,1]])
print "GCR4=", GCR4
CR4=LinearCode(GCR4)
print CR4
D4=CR4.minimum_distance()
print "D4=", D4
```

Dette giver outputtet

```
A=
[1 2 1 0]
[0 2 0 0]
[0 0 2 2]
[0 0 0 1]
```

```
GCR1= [1 2 1 0]
Linear code of length 4, dimension 1 over Finite Field of size 3
D1= 3
GCR2=
[1 2 1 0]
[0 2 0 0]
Linear code of length 4, dimension 2 over Finite Field of size 3
D2= 1
GCR3=
[1 2 1 0]
[0 2 0 0]
[0 0 2 2]
Linear code of length 4, dimension 3 over Finite Field of size 3
D3= 1
GCR4=
[1 2 1 0]
[0 2 0 0]
[0 0 2 2]
[0 0 0 1]
Linear code of length 4, dimension 4 over Finite Field of size 3
D4= 1
```

B.17 Kode til eksempel 4.2.1

```
#Nøglegenerering algoritme 1 punkt 2
#Find et primitivt element i F_17
k=GF(17)
k.primitive_element()
#Generer x_i'erne
x=[]
for i in range(1,17):
    x.append(3^(i-1)%17)
print "x=", x
#Generatormatrix for C_1
G1=matrix([[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],[x[0],x[1],x[2],x[3],x[4],x
    [5],x[6],x[7],x[8],x[9],x[10],x[11],x[12],x[13],x[14],x[15]],[x[0]^2,x
    [1]^2,x[2]^2,x[3]^2,x[4]^2,x[5]^2,x[6]^2,x[7]^2,x[8]^2,x[9]^2,x[10]^2,x
    [11]^2,x[12]^2,x[13]^2,x[14]^2,x[15]^2],[x[0]^3,x[1]^3,x[2]^3,x[3]^3,x
    [4]^3,x[5]^3,x[6]^3,x[7]^3,x[8]^3,x[9]^3,x[10]^3,x[11]^3,x[12]^3,x
    [13]^3,x[14]^3,x[15]^3],[x[0]^4,x[1]^4,x[2]^4,x[3]^4,x[4]^4,x[5]^4,x
    [6]^4,x[7]^4,x[8]^4,x[9]^4,x[10]^4,x[11]^4,x[12]^4,x[13]^4,x[14]^4,x
    [15]^4],[x[0]^5,x[1]^5,x[2]^5,x[3]^5,x[4]^5,x[5]^5,x[6]^5,x[7]^5,x
    [8]^5,x[9]^5,x[10]^5,x[11]^5,x[12]^5,x[13]^5,x[14]^5,x[15]^5],[x[0]^6,x
    [1]^6,x[2]^6,x[3]^6,x[4]^6,x[5]^6,x[6]^6,x[7]^6,x[8]^6,x[9]^6,x[10]^6,x
    [11]^6,x[12]^6,x[13]^6,x[14]^6,x[15]^6],[x[0]^7,x[1]^7,x[2]^7,x[3]^7,x
    [4]^7,x[5]^7,x[6]^7,x[7]^7,x[8]^7,x[9]^7,x[10]^7,x[11]^7,x[12]^7,x
    [13]^7,x[14]^7,x[15]^7],[x[0]^8,x[1]^8,x[2]^8,x[3]^8,x[4]^8,x[5]^8,x
    [6]^8,x[7]^8,x[8]^8,x[9]^8,x[10]^8,x[11]^8,x[12]^8,x[13]^8,x[14]^8,x
    [15]^8],[x[0]^9,x[1]^9,x[2]^9,x[3]^9,x[4]^9,x[5]^9,x[6]^9,x[7]^9,x
    [8]^9,x[9]^9,x[10]^9,x[11]^9,x[12]^9,x[13]^9,x[14]^9,x[15]^9]])%17
print "G1="
print G1
```

```
print ""
#Generatormatrix for C_2
G2=matrix([[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],[x[0],x[1],x[2],x[3],x[4],x
    [5],x[6],x[7],x[8],x[9],x[10],x[11],x[12],x[13],x[14],x[15]],[x[0]^2,x
    [1]^2,x[2]^2,x[3]^2,x[4]^2,x[5]^2,x[6]^2,x[7]^2,x[8]^2,x[9]^2,x[10]^2,x
    [11]^2,x[12]^2,x[13]^2,x[14]^2,x[15]^2],[x[0]^3,x[1]^3,x[2]^3,x[3]^3,x
    [4]^3,x[5]^3,x[6]^3,x[7]^3,x[8]^3,x[9]^3,x[10]^3,x[11]^3,x[12]^3,x
    [13]^3,x[14]^3,x[15]^3],[x[0]^4,x[1]^4,x[2]^4,x[3]^4,x[4]^4,x[5]^4,x
    [6]^4,x[7]^4,x[8]^4,x[9]^4,x[10]^4,x[11]^4,x[12]^4,x[13]^4,x[14]^4,x
    [15]^4],[x[0]^5,x[1]^5,x[2]^5,x[3]^5,x[4]^5,x[5]^5,x[6]^5,x[7]^5,x
    [8]^5,x[9]^5,x[10]^5,x[11]^5,x[12]^5,x[13]^5,x[14]^5,x[15]^5],[x[0]^6,x
    [1]^6,x[2]^6,x[3]^6,x[4]^6,x[5]^6,x[6]^6,x[7]^6,x[8]^6,x[9]^6,x[10]^6,x
    [11]^6,x[12]^6,x[13]^6,x[14]^6,x[15]^6],[x[0]^7,x[1]^7,x[2]^7,x[3]^7,x
    [4]^7,x[5]^7,x[6]^7,x[7]^7,x[8]^7,x[9]^7,x[10]^7,x[11]^7,x[12]^7,x
    [13]^7,x[14]^7,x[15]^7],[x[0]^8,x[1]^8,x[2]^8,x[3]^8,x[4]^8,x[5]^8,x
    [6]^8,x[7]^8,x[8]^8,x[9]^8,x[10]^8,x[11]^8,x[12]^8,x[13]^8,x[14]^8,x
    [15]^8]])%17
print "G2="
print G2
print ""
#Generatormatrix for C_3
G3=matrix([[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],[x[0],x[1],x[2],x[3],x[4],x
    [5],x[6],x[7],x[8],x[9],x[10],x[11],x[12],x[13],x[14],x[15]],[x[0]^2,x
    [1]^2,x[2]^2,x[3]^2,x[4]^2,x[5]^2,x[6]^2,x[7]^2,x[8]^2,x[9]^2,x[10]^2,x
    [11]^2,x[12]^2,x[13]^2,x[14]^2,x[15]^2],[x[0]^3,x[1]^3,x[2]^3,x[3]^3,x
    [4]^3,x[5]^3,x[6]^3,x[7]^3,x[8]^3,x[9]^3,x[10]^3,x[11]^3,x[12]^3,x
    [13]^3,x[14]^3,x[15]^3],[x[0]^4,x[1]^4,x[2]^4,x[3]^4,x[4]^4,x[5]^4,x
    [6]^4,x[7]^4,x[8]^4,x[9]^4,x[10]^4,x[11]^4,x[12]^4,x[13]^4,x[14]^4,x
    [15]^4],[x[0]^5,x[1]^5,x[2]^5,x[3]^5,x[4]^5,x[5]^5,x[6]^5,x[7]^5,x
    [8]^5,x[9]^5,x[10]^5,x[11]^5,x[12]^5,x[13]^5,x[14]^5,x[15]^5],[x[0]^6,x
    [1]^6,x[2]^6,x[3]^6,x[4]^6,x[5]^6,x[6]^6,x[7]^6,x[8]^6,x[9]^6,x[10]^6,x
    [11]^6,x[12]^6,x[13]^6,x[14]^6,x[15]^6],[x[0]^7,x[1]^7,x[2]^7,x[3]^7,x
    [4]^7,x[5]^7,x[6]^7,x[7]^7,x[8]^7,x[9]^7,x[10]^7,x[11]^7,x[12]^7,x
    [13]^7,x[14]^7,x[15]^7]])%17
print "G3="
print G3
print ""
A=matrix([[7,5,11],[0,10,9],[0,0,3]])
print"A="
print A
print""
A00G1=A[0,0]*G1%17
A01G1=A[0,1]*G1%17
A02G1=A[0,2]*G1%17
A10G2=A[1,0]*G2%17
A11G2=A[1,1]*G2%17
A12G2=A[1,2]*G2%17
A20G3=A[2,0]*G3%17
A21G3=A[2,1]*G3%17
A22G3=A[2,2]*G3%17
G=block_matrix([[A00G1,A01G1,A02G1],[A10G2,A11G2,A12G2],[A20G3,A21G3,A22G3
    ]])
```

```
39 print "G="
40 print G.str()
```

Dette giver outputtet

```
3
x= [1, 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6]
G1=
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
[ 1  3  9 10 13  5 15 11 16 14  8  7  4 12  2  6]
[ 1  9 13 15 16  8  4  2  1  9 13 15 16  8  4  2]
[ 1 10 15 14  4  6  9  5 16  7  2  3 13 11  8 12]
[ 1 13 16  4  1 13 16  4  1 13 16  4  1 13 16  4]
[ 1  5  8  6 13 14  2 10 16 12  9 11  4  3 15  7]
[ 1 15  4  9 16  2 13  8  1 15  4  9 16  2 13  8]
[ 1 11  2  5  4 10  8  3 16  6 15 12 13  7  9 14]
[ 1 16  1 16  1 16  1 16  1 16  1 16  1 16  1 16]
[ 1 14  9  7 13 12 15  6 16  3  8 10  4  5  2 11]

G2=
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
[ 1  3  9 10 13  5 15 11 16 14  8  7  4 12  2  6]
[ 1  9 13 15 16  8  4  2  1  9 13 15 16  8  4  2]
[ 1 10 15 14  4  6  9  5 16  7  2  3 13 11  8 12]
[ 1 13 16  4  1 13 16  4  1 13 16  4  1 13 16  4]
[ 1  5  8  6 13 14  2 10 16 12  9 11  4  3 15  7]
[ 1 15  4  9 16  2 13  8  1 15  4  9 16  2 13  8]
[ 1 11  2  5  4 10  8  3 16  6 15 12 13  7  9 14]
[ 1 16  1 16  1 16  1 16  1 16  1 16  1 16  1 16]

G3=
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
[ 1  3  9 10 13  5 15 11 16 14  8  7  4 12  2  6]
[ 1  9 13 15 16  8  4  2  1  9 13 15 16  8  4  2]
[ 1 10 15 14  4  6  9  5 16  7  2  3 13 11  8 12]
[ 1 13 16  4  1 13 16  4  1 13 16  4  1 13 16  4]
[ 1  5  8  6 13 14  2 10 16 12  9 11  4  3 15  7]
[ 1 15  4  9 16  2 13  8  1 15  4  9 16  2 13  8]
[ 1 11  2  5  4 10  8  3 16  6 15 12 13  7  9 14]

A=
[ 7  5 11]
[ 0 10  9]
[ 0  0  3]

G=
[ 7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7| 5  5  5  5  5  5  5  5
5  5  5  5  5  5  5  5|11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11]
[ 7  4 12  2  6  1  3  9 10 13  5 15 11 16 14  8| 5 15 11 16 14  8  7  4
 12  2  6  1  3  9 10 13|11 16 14  8  7  4 12  2  6  1  3  9 10 13  5 15]
[ 7 12  6  3 10  5 11 14  7 12  6  3 10  5 11 14| 5 11 14  7 12  6  3 10
  5 11 14  7 12  6  3 10|11 14  7 12  6  3 10  5 11 14  7 12  6  3 10  5]
[ 7  2  3 13 11  8 12  1 10 15 14  4  6  9  5 16| 5 16  7  2  3 13 11  8
```

```
 12   1 10 15 14   4   6   9|11   8 12   1 10 15 14   4   6   9   5 16   7   2   3 13]
[ 7   6 10 11   7   6 10 11   7   6 10 11   7   6 10 11| 5 14 12   3   5 14 12   3
  5 14 12   3   5 14 12   3|11   7   6 10 11   7   6 10 11   7   6 10 11   7   6 10]
[ 7   1   5   8   6 13 14   2 10 16 12   9 11   4   3 15| 5   8   6 13 14   2 10 16
 12   9 11   4   3 15   7   1|11   4   3 15   7   1   5   8   6 13 14   2 10 16 12   9]
[ 7   3 11 12 10 14   6   5   7   3 11 12 10 14   6   5| 5   7   3 11 12 10 14   6
  5   7   3 11 12 10 14   6|11 12 10 14   6   5   7   3 11 12 10 14   6   5   7   3]
[ 7   9 14   1 11   2   5   4 10   8   3 16   6 15 12 13| 5   4 10   8   3 16   6 15
 12 13   7   9 14   1 11   2|11   2   5   4 10   8   3 16   6 15 12 13   7   9 14   1]
[ 7 10   7 10   7 10   7 10   7 10   7 10   7 10   7 10| 5 12   5 12   5 12   5 12
  5 12   5 12   5 12   5 12|11   6 11   6 11   6 11   6 11   6 11   6 11   6 11   6]
[ 7 13 12 15   6 16   3   8 10   4   5   2 11   1 14   9| 5   2 11   1 14   9   7 13
 12 15   6 16   3   8 10   4|11   1 14   9   7 13 12 15   6 16   3   8 10   4   5   2]
[-----------------------------------------------+-------------------------
-----------------------+-----------------------------------------------]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0|10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10| 9   9   9   9   9   9   9   9   9   9   9   9   9   9   9   9]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0|10 13   5 15 11 16 14   8
 7   4 12   2   6   1   3   9| 9 10 13   5 15 11 16 14   8   7   4 12   2   6   1   3]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0|10   5 11 14   7 12   6   3
 10   5 11 14   7 12   6   3| 9 13 15 16   8   4   2   1   9 13 15 16   8   4   2   1]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0|10 15 14   4   6   9   5 16
  7   2   3 13 11   8 12   1| 9   5 16   7   2   3 13 11   8 12   1 10 15 14   4   6]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0|10 11   7   6 10 11   7   6
 10 11   7   6 10 11   7   6| 9 15   8   2   9 15   8   2   9 15   8   2   9 15   8   2]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0|10 16 12   9 11   4   3 15
  7   1   5   8   6 13 14   2| 9 11   4   3 15   7   1   5   8   6 13 14   2 10 16 12]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0|10 14   6   5   7   3 11 12
 10 14   6   5   7   3 11 12| 9 16   2 13   8   1 15   4   9 16   2 13   8   1 15   4]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0|10   8   3 16   6 15 12 13
  7   9 14   1 11   2   5   4| 9 14   1 11   2   5   4 10   8   3 16   6 15 12 13   7]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0|10   7 10   7 10   7 10   7
 10   7 10   7 10   7 10   7| 9   8   9   8   9   8   9   8   9   8   9   8   9   8   9   8]
[-----------------------------------------------+-------------------------
-----------------------+-----------------------------------------------]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0| 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0| 3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0| 0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0| 3   9 10 13   5 15 11 16 14   8   7   4 12   2   6   1]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0| 0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0| 3 10   5 11 14   7 12   6   3 10   5 11 14   7 12   6]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0| 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0| 3 13 11   8 12   1 10 15 14   4   6   9   5 16   7   2]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0| 0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0| 3   5 14 12   3   5 14 12   3   5 14 12   3   5 14 12]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0| 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0| 3 15   7   1   5   8   6 13 14   2 10 16 12   9 11   4]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0| 0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0| 3 11 12 10 14   6   5   7   3 11 12 10 14   6   5   7]
[ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0| 0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0| 3 16   6 15 12 13   7   9 14   1 11   2   5   4 10   8]
```

```
#Nøglegenerering algoritme 1 punkt 3
P=matrix([[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,1,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,1,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0],
```

```
0,0,0,0,0,0,0,1,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,1,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0],
[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0]])

#Nøglegenerering algoritme 1 punkt 4
S=matrix([[2,1,1,3,1,12,15,11,2,2,0,0,2,0,15,8,4,15,16,6,8,6,2,1,6,12,14],
[14,1,5,2,8,6,10,15,16,9,3,4,3,8,5,14,12,0,2,15,2,5,1,0,4,5,4],
[2,13,10,14,16,15,6,1,16,7,1,12,12,14,14,10,10,4,10,6,12,16,15,11,4,16,1],
[14,15,12,4,14,16,1,8,9,9,9,0,11,0,9,10,11,11,1,0,9,7,1,0,1,1,1],
[7,3,3,14,0,13,6,9,11,6,11,11,9,11,14,5,9,7,3,8,4,8,1,15,6,4,3],
```

```
[11,12,11,4,7,14,2,4,4,10,15,8,2,2,14,14,12,16,10,16,3,8,10,12,5,11,16],
[0,14,6,16,5,14,7,2,15,8,10,2,11,14,15,6,7,7,10,12,4,12,1,14,1,15,0],
[15,4,16,13,6,9,1,7,2,4,1,16,8,15,8,0,11,12,16,13,13,0,4,14,16,10,3],
[13,11,12,14,6,3,3,8,16,10,3,8,7,9,10,14,12,4,11,0,12,12,10,16,16,14,1],
[10,8,5,3,0,0,15,2,8,0,16,1,7,16,14,14,1,11,12,16,10,12,11,10,7,0,2],
[9,3,2,0,1,13,5,3,14,1,5,15,16,15,8,6,5,2,12,3,0,13,16,16,9,10,4],
[12,2,4,6,14,3,10,2,11,9,1,2,9,13,10,2,5,13,16,5,2,6,3,0,1,3,2],
[10,14,0,13,1,7,10,5,3,8,4,8,3,13,1,7,16,7,13,10,11,4,7,9,9,9,14],
[1,9,10,2,5,6,7,12,14,12,2,10,2,13,7,11,15,8,2,0,8,9,6,9,9,10,13],
[11,15,0,12,7,13,13,16,11,0,8,10,1,12,8,9,10,4,15,11,12,15,8,1,11,13,14],
[5,8,12,3,5,15,1,2,9,14,3,10,12,0,7,7,16,4,2,4,1,5,1,14,0,15,1],
[9,13,15,7,9,2,5,2,3,10,14,11,15,8,4,0,15,13,15,8,9,1,0,11,5,6,1],
[13,11,7,11,5,15,0,7,3,11,9,6,6,6,7,10,8,10,5,15,0,2,6,5,14,4,13],
[8,12,7,12,14,8,3,9,0,15,11,10,7,6,1,0,4,6,11,13,12,1,7,4,16,6,14],
[11,12,6,0,13,15,12,6,7,1,15,11,5,16,4,2,6,5,16,11,15,9,14,1,0,16,0],
[7,13,1,15,13,16,3,9,16,14,13,6,8,5,10,1,3,10,6,10,12,5,3,7,7,14,5],
[1,14,2,1,8,3,15,15,4,16,13,15,11,5,3,10,1,3,11,11,11,6,9,3,10,14,13],
[16,16,1,8,15,16,9,14,8,15,11,6,0,0,7,6,11,8,1,14,16,10,11,7,11,10,14],
[9,14,11,5,11,11,8,3,2,4,12,12,5,1,13,10,14,7,10,5,8,16,16,10,1,9,4],
[2,12,0,0,12,8,3,2,7,7,7,13,12,12,8,6,12,9,2,6,0,5,15,2,13,7,5],
[14,10,16,3,2,7,5,16,13,14,7,8,12,6,8,3,9,13,1,11,13,9,3,4,7,5,2],
[0,9,15,12,1,15,2,15,4,13,9,6,2,2,0,16,15,15,0,0,15,6,13,7,3,2,1]])

print "S er invertibel" , S.matrix_over_field().is_invertible()
```

Dette giver outputtet

```
S er invertibel True
```

```
#Nøglegenerering algoritme 1 punkt 5
Ghat=S*G*P%17
print "Ghat="
print Ghat.str()
```

Dette giver outputtet

```
Ghat=
[11 10  6  6 14  9  9 15 14 14 16 14  3 16  6  9 13 12 16  1  4  1 14 16  6
      2  3  3 15  7 14  4  7  0  3 10 12 14  3  2 15 13  8  2  1  7 15 14]
[ 4  7  2 16 14 14  7 10  4  2  0  9  1  6 15  9  4  4 11 11  8 13 14  0  0
     14 13 15  0  5  0  5 16  5  1  7 14  7  3 15  9  8 11 10  8 13  5  6]
[ 0  3  4  8 14  1 14  2 11  8  4  5  2  8 12 10 12  0 10  9  6  7  6  1 15
      6  4  4  6  6  1 10 14 10  3  2  5 14  0  6  0  2  0 16 16  5  2  4]
[14  0  6 12 14  9  9 12 12 13  3 13  3  7 10  8  6  3 13 10  5 11  9  3  3
      0  2 16  7  4  6  2  5  8 16  1 12 14 15  8 16 12 13  7 16  7  3 10]
[ 4 11 16  0 16  7  8  2  9  6  4 10 10  5  7  8  8 10  0  1  0  5 11  2 16
      1  8 13 13 11 10 12 16  4  1 16  5  6 11  3  2  7  5 16  3  8 13 11]
[ 4  9  4  6  2  2 15 15  8 10 16  7  2  9  6 16  0  5  3 10  3 14  7 11  4
     14  8  1 11 12 11  3 15 16 11 15  0 11 12  2  4  0 15 14 10  2  0  8]
[13 14  4  4  9 15  4 12  3  4 10 16  1 13  7  5 11  6 14 15 14 11  8  7  7
      9  7  5 15  3  2 14  0 14  0  7  3  3 11 15 15  8  2  2 16  4 16  5]
[13 12 15 12  4  3  9  6 11  7 16 14 10  5 11  3 10  4  6  3 16  8 10  3  8
      9 11  3 15  3 13  4  4 14 10  9 13 16 15 10  5 12 16  2  1 11  8  7]
[ 0  9 16  3  6  7  1 10 15 16  4 13 11 12  6  0  4 11  0 10 13 15 10  3  3
      9  9  0  9 15 12 14  8  2  2  4 12 13  6  3  3  5 14  1  5 13 15  3]
```

```
[ 2  0  2  5  4 12  6 11  1  4  3 13  4  1 14  4 13  5 12 12 12  1  0 16 14
  2  8  9 16 14 12  8 12  2  3 15  6  1 12 16  8  2  5  1 12 10  1  7]
[ 7  0  9  7  4  2  6  0  8 16  0  7  0  7 11  0  0  9  3 13 14 15  8 16 13
  1  2 10  0 14  0  7  6  7  9  8 12 12  4  3  1  8  2 13 12 12 16  1]
[ 8  1  5  4 14 16 10  9  6 10 12 12  5  1 11  3  9 16 11 11 12  4  9 16 11
  6  0 14 15 10 12  4 14  4  4  5 13  4 12 16  0  2 10  4 10  0  0  2]
[11  4 10  0 13  7  5 14 13  9  5 15 16 13 10  7  3  9  6  0  0  7 16  6 13
 13  7  1  3  3 16  9 10  4 11 15  9  9 13  4 11  7  1 10 16  9  7 12]
[ 7  2  6 16 10  0  4  5  7 14 11 15 12 11 10  4  4  6  5  8  3  6 13 15 13
 11 15  6  7  8 14  2 10  2 13  4  8 11 11  2  6  1 14 10  0  8 10  1]
[ 7  6 14 16  1  5  3 12 11 10 15  9  8  5 13  5  5  4  7 10 12 15  1 12 11
 10  9 14 11  4  3  7  4  2  6 12 12  4  9  5 16  7  3  3 12  3  0  5]
[ 8  8 13 11  4 12 12  3  4 13  8  4  0 13 13  0  6 15  2 15 14  8  7 11  2
  3 11 12 16 15  7  9 13 11  3  2  8  8 10  6  9  2  7 15 13 14  4  7]
[ 5 15  9 14 16  8 13  4 12  1  1  7 12 16 15 14  6 15  8  7 11  1 16 12 14
 13  9 16 11  8  1  7 12 16 15 16 12 10  6  5  7 13  1 10  8 14  5  5]
[ 7  3 14 12  9 11 15  8 13  8  2  7 15  2 15  3 16 15  5 16  1  6 10 12  0
  1  2  8  5  0 10  1  1 14 15 15  5  9 10  5  6 15  2 14 13 15 14 15]
[16  4  7 11 12 15 14 14  6 15  2  7  7 10  7  3 15  2 10 13 16  0 16 10  7
  0  3  2  2 11  8 14  1 14  6 16  6  5  6 13  2 14  6  8  6  6  0 13]
[13  3  4  4  1  8  9  5  3  2 11 12 15 10  8  9 10  3  3 16  4  7  2 15 13
 10  2  1  9 13 12  9  6  8  0  1  6 12  0  5  9 15 14 16  3 15  8  0]
[ 4  1  3  6 14  4 15  7  6 15 11 16  7  0 11 13 11 15  8 15  3 10 15 10  7
  8 13  6  2 11  3  1 10 16 10 15  4 13  9 14  7  8 15 11 14 12 11  6]
[10  9  7  7 10 13 10 12  1  6 11 16  0  4  5 15 15  3  9 15 14 10  0  1  7
  6 13  2  2  8 14  2 16 10 16  3  4  1 11 13  9 11  7  6  9  8 16 14]
[16 10 12  7  5  7 13  2 11 13  2  0  6 16  7 11  6 13 12  6 11  6 10 16  3
  2  5 10  2  2  4  5  4  2  6 13 16  9  1  9  2  7  4  8  5  4  0  9]
[11  2  1  2  0  6 15 16  1 11  5  5 16 10  4  0 10 11  0 13 16 13  7  4 14
 14  3 15 13  5  2 14  6  6 10  5  5 16 16  4  3 11 16  5 16 12  5  5]
[ 0 14 11  7 16  2 13  5  7  4 14  5  3  6  0 11  3 16 10  0  1  5 12  6 16
 10  2  0  1  5  9  9  0  4  5  4 13  1 15  8  1  8  7  0  2  1  2  2]
[ 8  3 16 10  9 15  7  2  9  9  9  6  3  4  7  0  4  0  3  0  6  8  0  0 10
  1  0  9  4 10 12 12  8 14 12 12  2 12 12  2 10  2  0  8  5  0  2 14]
[ 6  7  8 16  7  4  2 12  0 10 16  7  7 13 16 14 13 12  6  6  7  7  9  7 12
  9  2  2  6  9  6  1  7  9  0  4 10  2  5  6  5  4 14  6 10 13  0  2]
```

B.18 Kode til eksempel 4.2.2

```
#Indkodning, Algoritme 2
m=matrix
    ([[11,15,4,14,9,14,7,0,13,5,4,0,3,1,13,9,12,12,1,0,15,7,0,9,4,1,0]])
print "m=" , m
print ""
e=matrix
    ([[0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,15,
0,2,0,0,0,0,0,0,0,0,0,0,0,0,0]])
print "e=" , e
print ""
#algoritme 2 punkt 1
mGhat=m*Ghat%17
print "mGhat=" , mGhat
```

```
print ""
#algoritme 2 punkt 2
y=(mGhat+e)%17
print "y=" , y
```

Dette giver outputtet

```
e= [ 0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  7  0  0  0  0  0
     0  0  0  0  0  0  0  0 15  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0
    0]

mGhat= [ 8 16 14  4 11 14  0 14 16 14  5 11  6  6  1  1 11 16  7  4 14  9
    13  1 11 16  1  5 13  3  5  9  9  9 14 12 11  2  3  2  2 14 13 15 10  4
     16  3]

y= [ 8 16 14  4 11 14  0  0 16 14  5 11  6  6  1  1 11 16 14  4 14  9 13  1
     11 16  1  5 13  3  5  9  7  9 16 12 11  2  3  2  2 14 13 15 10  4 16
    3]
```

B.19 Kode til eksempel 4.2.4

```
#Dekodning, Algoritme 3
#algoritme 3 punkt 1
yhat=y*P^(-1)
print "yhat=" , yhat
```

Dette giver outputtet

```
yhat= [16  4 14  0 14 11  6  1 16  4  9  1 16  5  3  9  9 12  2  2 14 15  4
     3 16 10 13  2  3 11 16  7  5 13  1 11 13 14 14 11  1  6  5 16  0 11
    14  8]
```

```
#algoritme 3 punkt 2 (McEliece dekodning)
#Benyt kendt dekodningsalgoritme (algoritme 12) og herunder en kendt for
    hver af delkoderne.
#Cifferteksten deles i tre lige lange dele.
y1=matrix(GF(17),[[yhat[0,0],yhat[0,1],yhat[0,2],yhat[0,3],yhat[0,4],yhat
    [0,5],yhat[0,6],yhat[0,7],yhat[0,8],yhat[0,9],yhat[0,10],yhat[0,11],
    yhat[0,12],yhat[0,13],yhat[0,14],yhat[0,15]]])
print "y1=" , y1
print ""
y2=matrix(GF(17),[[yhat[0,16],yhat[0,17],yhat[0,18],yhat[0,19],yhat[0,20],
    yhat[0,21],yhat[0,22],yhat[0,23],yhat[0,24],yhat[0,25],yhat[0,26],yhat
    [0,27],yhat[0,28],yhat[0,29],yhat[0,30],yhat[0,31]]])
print "y2=" , y2
print ""
y3=matrix(GF(17),[[yhat[0,32],yhat[0,33],yhat[0,34],yhat[0,35],yhat[0,36],
    yhat[0,37],yhat[0,38],yhat[0,39],yhat[0,40],yhat[0,41],yhat[0,42],yhat
    [0,43],yhat[0,44],yhat[0,45],yhat[0,46],yhat[0,47]]])
print "y3=" , y3
```

Dette giver outputtet

```
y1= [16  4 14  0 14 11  6  1 16  4  9  1 16  5  3  9]

y2= [ 9 12  2  2 14 15  4  3 16 10 13  2  3 11 16  7]

y3= [ 5 13  1 11 13 14 14 11  1  6  5 16  0 11 14  8]
```

```
#Algoritme 12 punkt 2
#i1=1, i2=2 og i3=3
#Algoritme 12 punkt 4
#j=1
#Benyt dekodningsalgoritme DC1 på y1
#DC1 (algoritmen til dekodning af Reed-Solomon-koder fra appendiks A1)
ligsys= matrix(GF(17),[
[1,x[0],x[0]^2,x[0]^3,x[0]^4,x[0]^5,x[0]^6,x[0]^7,x[0]^8,x[0]^9,x[0]^(10),x
    [0]^(11),x[0]^(12),y1[0,0],y1[0,0]*x[0],y1[0,0]*x[0]^2,y1[0,0]*x[0]^3],
[1,x[1],x[1]^2,x[1]^3,x[1]^4,x[1]^5,x[1]^6,x[1]^7,x[1]^8,x[1]^9,x[1]^(10),x
    [1]^(11),x[1]^(12),y1[0,1],y1[0,1]*x[1],y1[0,1]*x[1]^2,y1[0,1]*x[1]^3],
[1,x[2],x[2]^2,x[2]^3,x[2]^4,x[2]^5,x[2]^6,x[2]^7,x[2]^8,x[2]^9,x[2]^(10),x
    [2]^(11),x[2]^(12),y1[0,2],y1[0,2]*x[2],y1[0,2]*x[2]^2,y1[0,2]*x[2]^3],
[1,x[3],x[3]^2,x[3]^3,x[3]^4,x[3]^5,x[3]^6,x[3]^7,x[3]^8,x[3]^9,x[3]^(10),x
    [3]^(11),x[3]^(12),y1[0,3],y1[0,3]*x[3],y1[0,3]*x[3]^2,y1[0,3]*x[3]^3],
[1,x[4],x[4]^2,x[4]^3,x[4]^4,x[4]^5,x[4]^6,x[4]^7,x[4]^8,x[4]^9,x[4]^(10),x
    [4]^(11),x[4]^(12),y1[0,4],y1[0,4]*x[4],y1[0,4]*x[4]^2,y1[0,4]*x[4]^3],
[1,x[5],x[5]^2,x[5]^3,x[5]^4,x[5]^5,x[5]^6,x[5]^7,x[5]^8,x[5]^9,x[5]^(10),x
    [5]^(11),x[5]^(12),y1[0,5],y1[0,5]*x[5],y1[0,5]*x[5]^2,y1[0,5]*x[5]^3],
[1,x[6],x[6]^2,x[6]^3,x[6]^4,x[6]^5,x[6]^6,x[6]^7,x[6]^8,x[6]^9,x[6]^(10),x
    [6]^(11),x[6]^(12),y1[0,6],y1[0,6]*x[6],y1[0,6]*x[6]^2,y1[0,6]*x[6]^3],
[1,x[7],x[7]^2,x[7]^3,x[7]^4,x[7]^5,x[7]^6,x[7]^7,x[7]^8,x[7]^9,x[7]^(10),x
    [7]^(11),x[7]^(12),y1[0,7],y1[0,7]*x[7],y1[0,7]*x[7]^2,y1[0,7]*x[7]^3],
[1,x[8],x[8]^2,x[8]^3,x[8]^4,x[8]^5,x[8]^6,x[8]^7,x[8]^8,x[8]^9,x[8]^(10),x
    [8]^(11),x[8]^(12),y1[0,8],y1[0,8]*x[8],y1[0,8]*x[8]^2,y1[0,8]*x[8]^3],
[1,x[9],x[9]^2,x[9]^3,x[9]^4,x[9]^5,x[9]^6,x[9]^7,x[9]^8,x[9]^9,x[9]^(10),x
    [9]^(11),x[9]^(12),y1[0,9],y1[0,9]*x[9],y1[0,9]*x[9]^2,y1[0,9]*x[9]^3],
[1,x[10],x[10]^2,x[10]^3,x[10]^4,x[10]^5,x[10]^6,x[10]^7,x[10]^8,x[10]^9,x
    [10]^(10),x[10]^(11),x[10]^(12),y1[0,10],y1[0,10]*x[10],y1[0,10]*x
    [10]^2,
y1[0,10]*x[10]^3],
[1,x[11],x[11]^2,x[11]^3,x[11]^4,x[11]^5,x[11]^6,x[11]^7,x[11]^8,x[11]^9,x
    [11]^(10),x[11]^(11),x[11]^(12),y1[0,11],y1[0,11]*x[11],y1[0,11]*x
    [11]^2,
y1[0,11]*x[11]^3],
[1,x[12],x[12]^2,x[12]^3,x[12]^4,x[12]^5,x[12]^6,x[12]^7,x[12]^8,x[12]^9,x
    [12]^(10),x[12]^(11),x[12]^(12),y1[0,12],y1[0,12]*x[12],y1[0,12]*x
    [12]^2,
y1[0,12]*x[12]^3],
[1,x[13],x[13]^2,x[13]^3,x[13]^4,x[13]^5,x[13]^6,x[13]^7,x[13]^8,x[13]^9,x
    [13]^(10),x[13]^(11),x[13]^(12),y1[0,13],y1[0,13]*x[13],y1[0,13]*x
    [13]^2,
y1[0,13]*x[13]^3],
[1,x[14],x[14]^2,x[14]^3,x[14]^4,x[14]^5,x[14]^6,x[14]^7,x[14]^8,x[14]^9,x
    [14]^(10),x[14]^(11),x[14]^(12),y1[0,14],y1[0,14]*x[14],y1[0,14]*x
```

```
    [14]^2,
y1[0,14]*x[14]^3],
[1,x[15],x[15]^2,x[15]^3,x[15]^4,x[15]^5,x[15]^6,x[15]^7,x[15]^8,x[15]^9,x
    [15]^(10),x[15]^(11),x[15]^(12),y1[0,15],y1[0,15]*x[15],y1[0,15]*x
    [15]^2,
y1[0,15]*x[15]^3]])
print "ligsys="
print ligsys
print ""
rre= ligsys.echelon_form()
print "ligningssystem på reduceret-række-echelon-form:"
print rre
```

Dette giver outputtet

```
ligsys=
[ 1  1  1  1  1  1  1  1  1  1  1  1  1 16 16 16 16]
[ 1  3  9 10 13  5 15 11 16 14  8  7  4  4 12  2  6]
[ 1  9 13 15 16  8  4  2  1  9 13 15 16 14  7 12  6]
[ 1 10 15 14  4  6  9  5 16  7  2  3 13  0  0  0  0]
[ 1 13 16  4  1 13 16  4  1 13 16  4  1 14 12  3  5]
[ 1  5  8  6 13 14  2 10 16 12  9 11  4 11  4  3 15]
[ 1 15  4  9 16  2 13  8  1 15  4  9 16  6  5  7  3]
[ 1 11  2  5  4 10  8  3 16  6 15 12 13  1 11  2  5]
[ 1 16  1 16  1 16  1 16  1 16  1 16  1 16  1 16  1]
[ 1 14  9  7 13 12 15  6 16  3  8 10  4  4  5  2 11]
[ 1  8 13  2 16  9  4 15  1  8 13  2 16  9  4 15  1]
[ 1  7 15  3  4 11  9 12 16 10  2 14 13  1  7 15  3]
[ 1  4 16 13  1  4 16 13  1  4 16 13  1 16 13  1  4]
[ 1 12  8 11 13  3  2  7 16  5  9  6  4  5  9  6  4]
[ 1  2  4  8 16 15 13  9  1  2  4  8 16  3  6 12  7]
[ 1  6  2 12  4  7  8 14 16 11 15  5 13  9  3  1  6]


ligningssystem på reduceret-række-echelon-form:
[ 1  0  0  0  0  0  0  0  0  0  0  0  0  0  2  3 13]
[ 0  1  0  0  0  0  0  0  0  0  0  0  0  0  8 14  4]
[ 0  0  1  0  0  0  0  0  0  0  0  0  0  0 11 16  9]
[ 0  0  0  1  0  0  0  0  0  0  0  0  0  0 12 12  9]
[ 0  0  0  0  1  0  0  0  0  0  0  0  0  0 16  2 14]
[ 0  0  0  0  0  1  0  0  0  0  0  0  0  0  1  9  0]
[ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  7  3 12]
[ 0  0  0  0  0  0  0  1  0  0  0  0  0  0 11 15 15]
[ 0  0  0  0  0  0  0  0  1  0  0  0  0  0 16  1  0]
[ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  1  9 16]
[ 0  0  0  0  0  0  0  0  0  0  1  0  0  0  9  6  8]
[ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  9  6]
[ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  9]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  1 10 15 14]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

```
x = PolynomialRing(GF(17),'x').gen()
Q0=4+13*x+8*x^2+8*x^3+3*x^4+0*x^5+5*x^6+2*x^7+0*x^8+1*x^9+9*x^(10)+11*x
    ^(11)+8*x^(12)
```

```
print "Q0=" , Q0
print ""
Q1=3+0*x+0*x^2+1*x^3
print "Q1=" , Q1
print ""
minusf = Q0.quo_rem(Q1)
print "-f =" , minusf
print ""
f=-(minusf[0])
print "f=" , f
print ""
y12=matrix([[f(1),f(3),f(9),f(10),f(13),f(5),f(15),f(11),f(16),f(14),f(8),f
    (7),f(4),f(12),f(2),f(6)]])
print "y12=" , y12
print ""
y22=y2-((5/7)%17)*y12 #k=2 punkt 9
print "y22=" , y22
print ""
y32=y3-((11/7)%17)*y12 #k=3 punkt 9
print "y32=" , y32
print ""
#den opdaterede A
A2=matrix([[7,0,0],[0,10,9],[0,0,3]])
print "A2="
print A2
```

Dette giver outputtet

```
Q0= 8*x^12 + 11*x^11 + 9*x^10 + x^9 + 2*x^7 + 5*x^6 + 3*x^4 + 8*x^3 + 8*x^2
     + 13*x + 4

Q1= x^3 + 3

-f = (8*x^9 + 11*x^8 + 9*x^7 + 11*x^6 + x^5 + 9*x^4 + 6*x^3 + 14*x^2 + 10*x
     + 7, 0)

f= 9*x^9 + 6*x^8 + 8*x^7 + 6*x^6 + 16*x^5 + 8*x^4 + 11*x^3 + 3*x^2 + 7*x +
    10

y12= [16  4 14 14 14 11  6  1 16  4  9  1 16  5  3  9]

y22= [ 0 14  9  9  4 12  7 12  7 12  9 11 11  5  9  3]

y32= [ 9 14 13  6  8  4  7  7  5  7  3 12  4  8  2  6]

A2=
[ 7  0  0]
[ 0 10  9]
[ 0  0  3]
```

```
#j=2
x=[]
for i in range(1,17):
```

```
    x.append(3^(i-1)%17)
#DC2 (algoritmen til dekodning af Reed-Solomon-koder fra appendiks A1)
ligsys= matrix(GF(17),[
[1,x[0],x[0]^2,x[0]^3,x[0]^4,x[0]^5,x[0]^6,x[0]^7,x[0]^8,x[0]^9,x[0]^(10),x
    [0]^(11),x[0]^(12),y22[0,0],y22[0,0]*x[0],y22[0,0]*x[0]^2,y22[0,0]*x
    [0]^3],
[1,x[1],x[1]^2,x[1]^3,x[1]^4,x[1]^5,x[1]^6,x[1]^7,x[1]^8,x[1]^9,x[1]^(10),x
    [1]^(11),x[1]^(12),y22[0,1],y22[0,1]*x[1],y22[0,1]*x[1]^2,y22[0,1]*x
    [1]^3],
[1,x[2],x[2]^2,x[2]^3,x[2]^4,x[2]^5,x[2]^6,x[2]^7,x[2]^8,x[2]^9,x[2]^(10),x
    [2]^(11),x[2]^(12),y22[0,2],y22[0,2]*x[2],y22[0,2]*x[2]^2,y22[0,2]*x
    [2]^3],
[1,x[3],x[3]^2,x[3]^3,x[3]^4,x[3]^5,x[3]^6,x[3]^7,x[3]^8,x[3]^9,x[3]^(10),x
    [3]^(11),x[3]^(12),y22[0,3],y22[0,3]*x[3],y22[0,3]*x[3]^2,y22[0,3]*x
    [3]^3],
[1,x[4],x[4]^2,x[4]^3,x[4]^4,x[4]^5,x[4]^6,x[4]^7,x[4]^8,x[4]^9,x[4]^(10),x
    [4]^(11),x[4]^(12),y22[0,4],y22[0,4]*x[4],y22[0,4]*x[4]^2,y22[0,4]*x
    [4]^3],
[1,x[5],x[5]^2,x[5]^3,x[5]^4,x[5]^5,x[5]^6,x[5]^7,x[5]^8,x[5]^9,x[5]^(10),x
    [5]^(11),x[5]^(12),y22[0,5],y22[0,5]*x[5],y22[0,5]*x[5]^2,y22[0,5]*x
    [5]^3],
[1,x[6],x[6]^2,x[6]^3,x[6]^4,x[6]^5,x[6]^6,x[6]^7,x[6]^8,x[6]^9,x[6]^(10),x
    [6]^(11),x[6]^(12),y22[0,6],y22[0,6]*x[6],y22[0,6]*x[6]^2,y22[0,6]*x
    [6]^3],
[1,x[7],x[7]^2,x[7]^3,x[7]^4,x[7]^5,x[7]^6,x[7]^7,x[7]^8,x[7]^9,x[7]^(10),x
    [7]^(11),x[7]^(12),y22[0,7],y22[0,7]*x[7],y22[0,7]*x[7]^2,y22[0,7]*x
    [7]^3],
[1,x[8],x[8]^2,x[8]^3,x[8]^4,x[8]^5,x[8]^6,x[8]^7,x[8]^8,x[8]^9,x[8]^(10),x
    [8]^(11),x[8]^(12),y22[0,8],y22[0,8]*x[8],y22[0,8]*x[8]^2,y22[0,8]*x
    [8]^3],
[1,x[9],x[9]^2,x[9]^3,x[9]^4,x[9]^5,x[9]^6,x[9]^7,x[9]^8,x[9]^9,x[9]^(10),x
    [9]^(11),x[9]^(12),y22[0,9],y22[0,9]*x[9],y22[0,9]*x[9]^2,y22[0,9]*x
    [9]^3],
[1,x[10],x[10]^2,x[10]^3,x[10]^4,x[10]^5,x[10]^6,x[10]^7,x[10]^8,x[10]^9,x
    [10]^(10),x[10]^(11),x[10]^(12),y22[0,10],y22[0,10]*x[10],y22[0,10]*x
    [10]^2,
y22[0,10]*x[10]^3],
[1,x[11],x[11]^2,x[11]^3,x[11]^4,x[11]^5,x[11]^6,x[11]^7,x[11]^8,x[11]^9,x
    [11]^(10),x[11]^(11),x[11]^(12),y22[0,11],y22[0,11]*x[11],y22[0,11]*x
    [11]^2,
y22[0,11]*x[11]^3],
[1,x[12],x[12]^2,x[12]^3,x[12]^4,x[12]^5,x[12]^6,x[12]^7,x[12]^8,x[12]^9,x
    [12]^(10),x[12]^(11),x[12]^(12),y22[0,12],y22[0,12]*x[12],y22[0,12]*x
    [12]^2,
y22[0,12]*x[12]^3],
[1,x[13],x[13]^2,x[13]^3,x[13]^4,x[13]^5,x[13]^6,x[13]^7,x[13]^8,x[13]^9,x
    [13]^(10),x[13]^(11),x[13]^(12),y22[0,13],y22[0,13]*x[13],y22[0,13]*x
    [13]^2,
y22[0,13]*x[13]^3],
[1,x[14],x[14]^2,x[14]^3,x[14]^4,x[14]^5,x[14]^6,x[14]^7,x[14]^8,x[14]^9,x
    [14]^(10),x[14]^(11),x[14]^(12),y22[0,14],y22[0,14]*x[14],y22[0,14]*x
    [14]^2,
```

```
y22[0,14]*x[14]^3],
[1,x[15],x[15]^2,x[15]^3,x[15]^4,x[15]^5,x[15]^6,x[15]^7,x[15]^8,x[15]^9,x
    [15]^(10),x[15]^(11),x[15]^(12),y22[0,15],y22[0,15]*x[15],y22[0,15]*x
    [15]^2,
y22[0,15]*x[15]^3]])
print "ligsys="
print ligsys
print ""
rre= ligsys.echelon_form()
print "ligningssystem på reduceret-række-echelon-form:"
print rre
```

Dette giver outputtet

```
ligsys=
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0]
[ 1  3  9 10 13  5 15 11 16 14  8  7  4 14  8  7  4]
[ 1  9 13 15 16  8  4  2  1  9 13 15 16  9 13 15 16]
[ 1 10 15 14  4  6  9  5 16  7  2  3 13  9  5 16  7]
[ 1 13 16  4  1 13 16  4  1 13 16  4  1  4  1 13 16]
[ 1  5  8  6 13 14  2 10 16 12  9 11  4 12  9 11  4]
[ 1 15  4  9 16  2 13  8  1 15  4  9 16  7  3 11 12]
[ 1 11  2  5  4 10  8  3 16  6 15 12 13 12 13  7  9]
[ 1 16  1 16  1 16  1 16  1 16  1 16  1  7 10  7 10]
[ 1 14  9  7 13 12 15  6 16  3  8 10  4 12 15  6 16]
[ 1  8 13  2 16  9  4 15  1  8 13  2 16  9  4 15  1]
[ 1  7 15  3  4 11  9 12 16 10  2 14 13 11  9 12 16]
[ 1  4 16 13  1  4 16 13  1  4 16 13  1 11 10  6  7]
[ 1 12  8 11 13  3  2  7 16  5  9  6  4  5  9  6  4]
[ 1  2  4  8 16 15 13  9  1  2  4  8 16  9  1  2  4]
[ 1  6  2 12  4  7  8 14 16 11 15  5 13  3  1  6  2]

ligningssystem på reduceret-række-echelon-form:
[ 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  7  5]
[ 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  8  3]
[ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0 15  9]
[ 0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0 15]
[ 0  0  0  0  1  0  0  0  0  0  0  0  0  0  0 16  9]
[ 0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  9  3]
[ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  2  8]
[ 0  0  0  0  0  0  0  1  0  0  0  0  0  0  0 15  3]
[ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0 16  7]
[ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  5  5]
[ 0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  9  9]
[ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  9]
[ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  5  6]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  8  1]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

```
x = PolynomialRing(GF(17),'x').gen()
Q0=12+14*x+8*x^2+2*x^3+8*x^4+14*x^5+9*x^6+14*x^7+10*x^8+12*x^9+8*x^(10)+8*x
    ^(11)+0*x^(12)
```

```
print "Q0=" , Q0
print ""
Q1=6+1*x+0*x^2+1*x^3
print "Q1=" , Q1
print ""
minusf = Q0.quo_rem(Q1)
print "-f =" , minusf
```

dette giver outputtet

```
Q0= 8*x^11 + 8*x^10 + 12*x^9 + 10*x^8 + 14*x^7 + 9*x^6 + 14*x^5 + 8*x^4 +
    2*x^3 + 8*x^2 + 14*x + 12

Q1= x^3 + x + 6

-f = (8*x^8 + 8*x^7 + 4*x^6 + 5*x^5 + 13*x^4 + 14*x^3 + 5*x^2 + x + 15, 11*
    x^2 + 10*x + 7)
```

Da der er en rest, vælges en ny indeksmængde, $i1 = 1$, $i2 = 3$ og $i3 = 2$, men også her bliver der en rest ved brug af DC2. Derfor forsøges følgende.

```
#i1=2, i2=1 og i3=3
#j=1
x=[]
for i in range(1,17):
    x.append(3^(i-1)%17)
#DC1 (algoritmen til dekodning af Reed-Solomon-koder fra appendiks A1)
ligsys= matrix(GF(17),[
[1,x[0],x[0]^2,x[0]^3,x[0]^4,x[0]^5,x[0]^6,x[0]^7,x[0]^8,x[0]^9,x[0]^(10),x
    [0]^(11),x[0]^(12),y2[0,0],y2[0,0]*x[0],y2[0,0]*x[0]^2,y2[0,0]*x[0]^3],
[1,x[1],x[1]^2,x[1]^3,x[1]^4,x[1]^5,x[1]^6,x[1]^7,x[1]^8,x[1]^9,x[1]^(10),x
    [1]^(11),x[1]^(12),y2[0,1],y2[0,1]*x[1],y2[0,1]*x[1]^2,y2[0,1]*x[1]^3],
[1,x[2],x[2]^2,x[2]^3,x[2]^4,x[2]^5,x[2]^6,x[2]^7,x[2]^8,x[2]^9,x[2]^(10),x
    [2]^(11),x[2]^(12),y2[0,2],y2[0,2]*x[2],y2[0,2]*x[2]^2,y2[0,2]*x[2]^3],
[1,x[3],x[3]^2,x[3]^3,x[3]^4,x[3]^5,x[3]^6,x[3]^7,x[3]^8,x[3]^9,x[3]^(10),x
    [3]^(11),x[3]^(12),y2[0,3],y2[0,3]*x[3],y2[0,3]*x[3]^2,y2[0,3]*x[3]^3],
[1,x[4],x[4]^2,x[4]^3,x[4]^4,x[4]^5,x[4]^6,x[4]^7,x[4]^8,x[4]^9,x[4]^(10),x
    [4]^(11),x[4]^(12),y2[0,4],y2[0,4]*x[4],y2[0,4]*x[4]^2,y2[0,4]*x[4]^3],
[1,x[5],x[5]^2,x[5]^3,x[5]^4,x[5]^5,x[5]^6,x[5]^7,x[5]^8,x[5]^9,x[5]^(10),x
    [5]^(11),x[5]^(12),y2[0,5],y2[0,5]*x[5],y2[0,5]*x[5]^2,y2[0,5]*x[5]^3],
[1,x[6],x[6]^2,x[6]^3,x[6]^4,x[6]^5,x[6]^6,x[6]^7,x[6]^8,x[6]^9,x[6]^(10),x
    [6]^(11),x[6]^(12),y2[0,6],y2[0,6]*x[6],y2[0,6]*x[6]^2,y2[0,6]*x[6]^3],
[1,x[7],x[7]^2,x[7]^3,x[7]^4,x[7]^5,x[7]^6,x[7]^7,x[7]^8,x[7]^9,x[7]^(10),x
    [7]^(11),x[7]^(12),y2[0,7],y2[0,7]*x[7],y2[0,7]*x[7]^2,y2[0,7]*x[7]^3],
[1,x[8],x[8]^2,x[8]^3,x[8]^4,x[8]^5,x[8]^6,x[8]^7,x[8]^8,x[8]^9,x[8]^(10),x
    [8]^(11),x[8]^(12),y2[0,8],y2[0,8]*x[8],y2[0,8]*x[8]^2,y2[0,8]*x[8]^3],
[1,x[9],x[9]^2,x[9]^3,x[9]^4,x[9]^5,x[9]^6,x[9]^7,x[9]^8,x[9]^9,x[9]^(10),x
    [9]^(11),x[9]^(12),y2[0,9],y2[0,9]*x[9],y2[0,9]*x[9]^2,y2[0,9]*x[9]^3],
[1,x[10],x[10]^2,x[10]^3,x[10]^4,x[10]^5,x[10]^6,x[10]^7,x[10]^8,x[10]^9,x
    [10]^(10),x[10]^(11),x[10]^(12),y2[0,10],y2[0,10]*x[10],y2[0,10]*x
    [10]^2,
y2[0,10]*x[10]^3],
[1,x[11],x[11]^2,x[11]^3,x[11]^4,x[11]^5,x[11]^6,x[11]^7,x[11]^8,x[11]^9,x
    [11]^(10),x[11]^(11),x[11]^(12),y2[0,11],y2[0,11]*x[11],y2[0,11]*x
```

```
    [11]^2,
y2[0,11]*x[11]^3],
[1,x[12],x[12]^2,x[12]^3,x[12]^4,x[12]^5,x[12]^6,x[12]^7,x[12]^8,x[12]^9,x
    [12]^(10),x[12]^(11),x[12]^(12),y2[0,12],y2[0,12]*x[12],y2[0,12]*x
    [12]^2,
y2[0,12]*x[12]^3],
[1,x[13],x[13]^2,x[13]^3,x[13]^4,x[13]^5,x[13]^6,x[13]^7,x[13]^8,x[13]^9,x
    [13]^(10),x[13]^(11),x[13]^(12),y2[0,13],y2[0,13]*x[13],y2[0,13]*x
    [13]^2,
y2[0,13]*x[13]^3],
[1,x[14],x[14]^2,x[14]^3,x[14]^4,x[14]^5,x[14]^6,x[14]^7,x[14]^8,x[14]^9,x
    [14]^(10),x[14]^(11),x[14]^(12),y2[0,14],y2[0,14]*x[14],y2[0,14]*x
    [14]^2,
y2[0,14]*x[14]^3],
[1,x[15],x[15]^2,x[15]^3,x[15]^4,x[15]^5,x[15]^6,x[15]^7,x[15]^8,x[15]^9,x
    [15]^(10),x[15]^(11),x[15]^(12),y2[0,15],y2[0,15]*x[15],y2[0,15]*x
    [15]^2,
y2[0,15]*x[15]^3]])
print "ligsys="
print ligsys
print ""
rre= ligsys.echelon_form()
print "ligningssystem på reduceret-række-echelon-form:"
print rre
```

Dette giver outputtet

```
ligsys=
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  9  9  9  9]
[ 1  3  9 10 13  5 15 11 16 14  8  7  4 12  2  6  1]
[ 1  9 13 15 16  8  4  2  1  9 13 15 16  2  1  9 13]
[ 1 10 15 14  4  6  9  5 16  7  2  3 13  2  3 13 11]
[ 1 13 16  4  1 13 16  4  1 13 16  4  1 14 12  3  5]
[ 1  5  8  6 13 14  2 10 16 12  9 11  4 15  7  1  5]
[ 1 15  4  9 16  2 13  8  1 15  4  9 16  4  9 16  2]
[ 1 11  2  5  4 10  8  3 16  6 15 12 13  3 16  6 15]
[ 1 16  1 16  1 16  1 16  1 16  1 16  1 16  1 16  1]
[ 1 14  9  7 13 12 15  6 16  3  8 10  4 10  4  5  2]
[ 1  8 13  2 16  9  4 15  1  8 13  2 16 13  2 16  9]
[ 1  7 15  3  4 11  9 12 16 10  2 14 13  2 14 13  6]
[ 1  4 16 13  1  4 16 13  1  4 16 13  1  3 12 14  5]
[ 1 12  8 11 13  3  2  7 16  5  9  6  4 11 13  3  2]
[ 1  2  4  8 16 15 13  9  1  2  4  8 16 16 15 13  9]
[ 1  6  2 12  4  7  8 14 16 11 15  5 13  7  8 14 16]

ligningssystem på reduceret-række-echelon-form:
[ 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0 15  1]
[ 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  6 12]
[ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  3 13]
[ 0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  2  2]
[ 0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  2  1]
[ 0  0  0  0  0  1  0  0  0  0  0  0  0  0  0 16 11]
[ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  9  3]
```

```
[ 0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  9]
[ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  9  4]
[ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  5 15]
[ 0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  8  1]
[ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  4  6]
[ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  4]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  5  6]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  8  1]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

```
x = PolynomialRing(GF(17),'x').gen()
Q0=16+5*x+4*x^2+15*x^3+16*x^4+6*x^5+14*x^6+8*x^7+13*x^8+2*x^9+16*x^(10)+11*
    x^(11)+13*x^(12)
print "Q0=" , Q0
print ""
Q1=11+16*x+0*x^2+x^3
print "Q1=" , Q1
print ""
minusf = Q0.quo_rem(Q1)
print "-f =" , minusf
print ""
f=-(minusf[0])
print "f=" , f
print ""
y22=matrix([[f(1),f(3),f(9),f(10),f(13),f(5),f(15),f(11),f(16),f(14),f(8),f
    (7),f(4),f(12),f(2),f(6)]])
print "y22=" , y22
print ""
y12=y1-((7/5)%17)*y22
print "y12=" , y12
print ""
y32=y3-((11/5)%17)*y22
print "y32=" , y32
print ""
#den opdaterede A
A2=matrix([[0,5,0],[3,10,4],[0,0,3]])
print "A2="
print A2
```

Dette giver outputtet

```
Q0= 13*x^12 + 11*x^11 + 16*x^10 + 2*x^9 + 13*x^8 + 8*x^7 + 14*x^6 + 6*x^5 +
     16*x^4 + 15*x^3 + 4*x^2 + 5*x + 16

Q1= x^3 + 16*x + 11

-f = (13*x^9 + 11*x^8 + 12*x^7 + 6*x^6 + 6*x^5 + x^4 + 5*x^3 + 9*x^2 + 10*x
     + 3, 0)

f= 4*x^9 + 6*x^8 + 5*x^7 + 11*x^6 + 11*x^5 + 16*x^4 + 12*x^3 + 8*x^2 + 7*x
    + 14

y22= [ 9 12  2  2 14 15  4  3 16 10 13  2  3 11 14  9]
```

```
y12= [ 0 11  1  4  8  7 14  7 14  7  1  5  5 10 14 10]

y32= [ 9  7  0 10  6 15 12  1 10  1  7 15  7 14  7 12]

A2=
[ 0  5  0]
[ 3 10  4]
[ 0  0  3]
```

```
#j=2
x=[]
for i in range(1,17):
    x.append(3^(i-1)%17)
#DC2 (algoritmen til dekodning af Reed-Solomon-koder fra appendiks A1)
ligsys= matrix(GF(17),[
[1,x[0],x[0]^2,x[0]^3,x[0]^4,x[0]^5,x[0]^6,x[0]^7,x[0]^8,x[0]^9,x[0]^(10),x
    [0]^(11),x[0]^(12),y12[0,0],y12[0,0]*x[0],y12[0,0]*x[0]^2,y12[0,0]*x
    [0]^3],
[1,x[1],x[1]^2,x[1]^3,x[1]^4,x[1]^5,x[1]^6,x[1]^7,x[1]^8,x[1]^9,x[1]^(10),x
    [1]^(11),x[1]^(12),y12[0,1],y12[0,1]*x[1],y12[0,1]*x[1]^2,y12[0,1]*x
    [1]^3],
[1,x[2],x[2]^2,x[2]^3,x[2]^4,x[2]^5,x[2]^6,x[2]^7,x[2]^8,x[2]^9,x[2]^(10),x
    [2]^(11),x[2]^(12),y12[0,2],y12[0,2]*x[2],y12[0,2]*x[2]^2,y12[0,2]*x
    [2]^3],
[1,x[3],x[3]^2,x[3]^3,x[3]^4,x[3]^5,x[3]^6,x[3]^7,x[3]^8,x[3]^9,x[3]^(10),x
    [3]^(11),x[3]^(12),y12[0,3],y12[0,3]*x[3],y12[0,3]*x[3]^2,y12[0,3]*x
    [3]^3],
[1,x[4],x[4]^2,x[4]^3,x[4]^4,x[4]^5,x[4]^6,x[4]^7,x[4]^8,x[4]^9,x[4]^(10),x
    [4]^(11),x[4]^(12),y12[0,4],y12[0,4]*x[4],y12[0,4]*x[4]^2,y12[0,4]*x
    [4]^3],
[1,x[5],x[5]^2,x[5]^3,x[5]^4,x[5]^5,x[5]^6,x[5]^7,x[5]^8,x[5]^9,x[5]^(10),x
    [5]^(11),x[5]^(12),y12[0,5],y12[0,5]*x[5],y12[0,5]*x[5]^2,y12[0,5]*x
    [5]^3],
[1,x[6],x[6]^2,x[6]^3,x[6]^4,x[6]^5,x[6]^6,x[6]^7,x[6]^8,x[6]^9,x[6]^(10),x
    [6]^(11),x[6]^(12),y12[0,6],y12[0,6]*x[6],y12[0,6]*x[6]^2,y12[0,6]*x
    [6]^3],
[1,x[7],x[7]^2,x[7]^3,x[7]^4,x[7]^5,x[7]^6,x[7]^7,x[7]^8,x[7]^9,x[7]^(10),x
    [7]^(11),x[7]^(12),y12[0,7],y12[0,7]*x[7],y12[0,7]*x[7]^2,y12[0,7]*x
    [7]^3],
[1,x[8],x[8]^2,x[8]^3,x[8]^4,x[8]^5,x[8]^6,x[8]^7,x[8]^8,x[8]^9,x[8]^(10),x
    [8]^(11),x[8]^(12),y12[0,8],y12[0,8]*x[8],y12[0,8]*x[8]^2,y12[0,8]*x
    [8]^3],
[1,x[9],x[9]^2,x[9]^3,x[9]^4,x[9]^5,x[9]^6,x[9]^7,x[9]^8,x[9]^9,x[9]^(10),x
    [9]^(11),x[9]^(12),y12[0,9],y12[0,9]*x[9],y12[0,9]*x[9]^2,y12[0,9]*x
    [9]^3],
[1,x[10],x[10]^2,x[10]^3,x[10]^4,x[10]^5,x[10]^6,x[10]^7,x[10]^8,x[10]^9,x
    [10]^(10),x[10]^(11),x[10]^(12),y12[0,10],y12[0,10]*x[10],y12[0,10]*x
    [10]^2,
y12[0,10]*x[10]^3],
[1,x[11],x[11]^2,x[11]^3,x[11]^4,x[11]^5,x[11]^6,x[11]^7,x[11]^8,x[11]^9,x
    [11]^(10),x[11]^(11),x[11]^(12),y12[0,11],y12[0,11]*x[11],y12[0,11]*x
```

```
    [11]^2,
y12[0,11]*x[11]^3],
[1,x[12],x[12]^2,x[12]^3,x[12]^4,x[12]^5,x[12]^6,x[12]^7,x[12]^8,x[12]^9,x
    [12]^(10),x[12]^(11),x[12]^(12),y12[0,12],y12[0,12]*x[12],y12[0,12]*x
    [12]^2,
y12[0,12]*x[12]^3],
[1,x[13],x[13]^2,x[13]^3,x[13]^4,x[13]^5,x[13]^6,x[13]^7,x[13]^8,x[13]^9,x
    [13]^(10),x[13]^(11),x[13]^(12),y12[0,13],y12[0,13]*x[13],y12[0,13]*x
    [13]^2,
y12[0,13]*x[13]^3],
[1,x[14],x[14]^2,x[14]^3,x[14]^4,x[14]^5,x[14]^6,x[14]^7,x[14]^8,x[14]^9,x
    [14]^(10),x[14]^(11),x[14]^(12),y12[0,14],y12[0,14]*x[14],y12[0,14]*x
    [14]^2,
y12[0,14]*x[14]^3],
[1,x[15],x[15]^2,x[15]^3,x[15]^4,x[15]^5,x[15]^6,x[15]^7,x[15]^8,x[15]^9,x
    [15]^(10),x[15]^(11),x[15]^(12),y12[0,15],y12[0,15]*x[15],y12[0,15]*x
    [15]^2,
y12[0,15]*x[15]^3]])
print "ligsys="
print ligsys
print ""
rre= ligsys.echelon_form()
print "ligningssystem på reduceret-række-echelon-form:"
print rre
```

Dette giver outputtet

```
ligsys=
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0]
[ 1  3  9 10 13  5 15 11 16 14  8  7  4 11 16 14  8]
[ 1  9 13 15 16  8  4  2  1  9 13 15 16  1  9 13 15]
[ 1 10 15 14  4  6  9  5 16  7  2  3 13  4  6  9  5]
[ 1 13 16  4  1 13 16  4  1 13 16  4  1  8  2  9 15]
[ 1  5  8  6 13 14  2 10 16 12  9 11  4  7  1  5  8]
[ 1 15  4  9 16  2 13  8  1 15  4  9 16 14  6  5  7]
[ 1 11  2  5  4 10  8  3 16  6 15 12 13  7  9 14  1]
[ 1 16  1 16  1 16  1 16  1 16  1 16  1 14  3 14  3]
[ 1 14  9  7 13 12 15  6 16  3  8 10  4  7 13 12 15]
[ 1  8 13  2 16  9  4 15  1  8 13  2 16  1  8 13  2]
[ 1  7 15  3  4 11  9 12 16 10  2 14 13  5  1  7 15]
[ 1  4 16 13  1  4 16 13  1  4 16 13  1  5  3 12 14]
[ 1 12  8 11 13  3  2  7 16  5  9  6  4 10  1 12  8]
[ 1  2  4  8 16 15 13  9  1  2  4  8 16 14 11  5 10]
[ 1  6  2 12  4  7  8 14 16 11 15  5 13 10  9  3  1]

ligningssystem på reduceret-række-echelon-form:
[ 1  0  0  0  0  0  0  0  0  0  0  0  0  0 11  8 12]
[ 0  1  0  0  0  0  0  0  0  0  0  0  0  0 15  8 12]
[ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  8  6]
[ 0  0  0  1  0  0  0  0  0  0  0  0  0  0  9  6  7]
[ 0  0  0  0  1  0  0  0  0  0  0  0  0  0  9 14  5]
[ 0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  9 14]
[ 0  0  0  0  0  0  1  0  0  0  0  0  0  0 13 11  0]
```

```
[ 0  0  0  0  0  0  0  1  0  0  0  0  0  0  1  6  9]
[ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  8 13  7]
[ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  1  1 11]
[ 0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  1  1]
[ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  1]
[ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  1 10 15 14]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

```
x = PolynomialRing(GF(17),'x').gen()
Q0=5+5*x+11*x^2+10*x^3+12*x^4+3*x^5+0*x^6+8*x^7+10*x^8+6*x^9+16*x^(10)+16*x^(11)+0*x^(12)
print "Q0=" , Q0
print ""
Q1=3+0*x+0*x^2+1*x^3
print "Q1=" , Q1
print ""
minusf = Q0.quo_rem(Q1)
print "-f =" , minusf
print ""
f=-(minusf[0])
print "f=" , f
print ""
y13=matrix([[f(1),f(3),f(9),f(10),f(13),f(5),f(15),f(11),f(16),f(14),f(8),f(7),f(4),f(12),f(2),f(6)]])
print "y13=" , y13
print ""
y33=y32-((4/3)%17)*y13
print "y33=" , y33
print ""
#den opdaterede A
A3=matrix([[0,5,0],[3,10,0],[0,0,3]])
print "A3="
print A3
```

Dette giver outputtet

```
Q0= 16*x^11 + 16*x^10 + 6*x^9 + 10*x^8 + 8*x^7 + 3*x^5 + 12*x^4 + 10*x^3 + 11*x^2 + 5*x + 5

Q1= x^3 + 3

-f = (16*x^8 + 16*x^7 + 6*x^6 + 13*x^5 + 11*x^4 + 16*x^3 + 15*x^2 + 13*x + 13, 0)

f= x^8 + x^7 + 11*x^6 + 4*x^5 + 6*x^4 + x^3 + 2*x^2 + 4*x + 4

y13= [ 0 11  1  1  8  7 14  7 14  7  1  5  5 10 14 10]

y33= [ 9 15 10  3  1  0 16  3 14  3  0 14  6 12 11 10]
```

```
A3=
[ 0  5  0]
[ 3 10  0]
[ 0  0  3]
```

```
#j=3
x=[]
for i in range(1,17):
    x.append(3^(i-1)%17)
#DC3 algoritmen til dekodning af Reed-Solomon-koder fra appendiks A1
ligsys= matrix(GF(17),[
[1,x[0],x[0]^2,x[0]^3,x[0]^4,x[0]^5,x[0]^6,x[0]^7,x[0]^8,x[0]^9,x[0]^(10),x
    [0]^(11),y33[0,0],y33[0,0]*x[0],y33[0,0]*x[0]^2,y33[0,0]*x[0]^3,y33
    [0,0]*x[0]^4],
[1,x[1],x[1]^2,x[1]^3,x[1]^4,x[1]^5,x[1]^6,x[1]^7,x[1]^8,x[1]^9,x[1]^(10),x
    [1]^(11),y33[0,1],y33[0,1]*x[1],y33[0,1]*x[1]^2,y33[0,1]*x[1]^3,y33
    [0,1]*x[1]^4],
[1,x[2],x[2]^2,x[2]^3,x[2]^4,x[2]^5,x[2]^6,x[2]^7,x[2]^8,x[2]^9,x[2]^(10),x
    [2]^(11),y33[0,2],y33[0,2]*x[2],y33[0,2]*x[2]^2,y33[0,2]*x[2]^3,y33
    [0,2]*x[2]^4],
[1,x[3],x[3]^2,x[3]^3,x[3]^4,x[3]^5,x[3]^6,x[3]^7,x[3]^8,x[3]^9,x[3]^(10),x
    [3]^(11),y33[0,3],y33[0,3]*x[3],y33[0,3]*x[3]^2,y33[0,3]*x[3]^3,y33
    [0,3]*x[3]^4],
[1,x[4],x[4]^2,x[4]^3,x[4]^4,x[4]^5,x[4]^6,x[4]^7,x[4]^8,x[4]^9,x[4]^(10),x
    [4]^(11),y33[0,4],y33[0,4]*x[4],y33[0,4]*x[4]^2,y33[0,4]*x[4]^3,y33
    [0,4]*x[4]^4],
[1,x[5],x[5]^2,x[5]^3,x[5]^4,x[5]^5,x[5]^6,x[5]^7,x[5]^8,x[5]^9,x[5]^(10),x
    [5]^(11),y33[0,5],y33[0,5]*x[5],y33[0,5]*x[5]^2,y33[0,5]*x[5]^3,y33
    [0,5]*x[5]^4],
[1,x[6],x[6]^2,x[6]^3,x[6]^4,x[6]^5,x[6]^6,x[6]^7,x[6]^8,x[6]^9,x[6]^(10),x
    [6]^(11),y33[0,6],y33[0,6]*x[6],y33[0,6]*x[6]^2,y33[0,6]*x[6]^3,y33
    [0,6]*x[6]^4],
[1,x[7],x[7]^2,x[7]^3,x[7]^4,x[7]^5,x[7]^6,x[7]^7,x[7]^8,x[7]^9,x[7]^(10),x
    [7]^(11),y33[0,7],y33[0,7]*x[7],y33[0,7]*x[7]^2,y33[0,7]*x[7]^3,y33
    [0,7]*x[7]^4],
[1,x[8],x[8]^2,x[8]^3,x[8]^4,x[8]^5,x[8]^6,x[8]^7,x[8]^8,x[8]^9,x[8]^(10),x
    [8]^(11),y33[0,8],y33[0,8]*x[8],y33[0,8]*x[8]^2,y33[0,8]*x[8]^3,y33
    [0,8]*x[8]^4],
[1,x[9],x[9]^2,x[9]^3,x[9]^4,x[9]^5,x[9]^6,x[9]^7,x[9]^8,x[9]^9,x[9]^(10),x
    [9]^(11),y33[0,9],y33[0,9]*x[9],y33[0,9]*x[9]^2,y33[0,9]*x[9]^3,y33
    [0,9]*x[9]^4],
[1,x[10],x[10]^2,x[10]^3,x[10]^4,x[10]^5,x[10]^6,x[10]^7,x[10]^8,x[10]^9,x
    [10]^(10),x[10]^(11),y33[0,10],y33[0,10]*x[10],y33[0,10]*x[10]^2,
y33[0,10]*x[10]^3,y33[0,10]*x[10]^4],
[1,x[11],x[11]^2,x[11]^3,x[11]^4,x[11]^5,x[11]^6,x[11]^7,x[11]^8,x[11]^9,x
    [11]^(10),x[11]^(11),y33[0,11],y33[0,11]*x[11],y33[0,11]*x[11]^2,
y33[0,11]*x[11]^3,y33[0,11]*x[11]^4],
[1,x[12],x[12]^2,x[12]^3,x[12]^4,x[12]^5,x[12]^6,x[12]^7,x[12]^8,x[12]^9,x
    [12]^(10),x[12]^(11),y33[0,12],y33[0,12]*x[12],y33[0,12]*x[12]^2,
y33[0,12]*x[12]^3,y33[0,12]*x[12]^4],
[1,x[13],x[13]^2,x[13]^3,x[13]^4,x[13]^5,x[13]^6,x[13]^7,x[13]^8,x[13]^9,x
    [13]^(10),x[13]^(11),y33[0,13],y33[0,13]*x[13],y33[0,13]*x[13]^2,
```

```
y33[0,13]*x[13]^3,y33[0,13]*x[13]^4],
[1,x[14],x[14]^2,x[14]^3,x[14]^4,x[14]^5,x[14]^6,x[14]^7,x[14]^8,x[14]^9,x[14]^(10),x[14]^(11),y33[0,14],y33[0,14]*x[14],y33[0,14]*x[14]^2,
y33[0,14]*x[14]^3,y33[0,14]*x[14]^4],
[1,x[15],x[15]^2,x[15]^3,x[15]^4,x[15]^5,x[15]^6,x[15]^7,x[15]^8,x[15]^9,x[15]^(10),x[15]^(11),y33[0,15],y33[0,15]*x[15],y33[0,15]*x[15]^2,
y33[0,15]*x[15]^3,y33[0,15]*x[15]^4]])
print "ligsys="
print ligsys
print ""
rre= ligsys.echelon_form()
print "ligningssystem på reduceret-række-echelon-form:"
print rre
```

Dette giver outputtet

```
ligsys=
[ 1  1  1  1  1  1  1  1  1  1  1  1  9  9  9  9  9]
[ 1  3  9 10 13  5 15 11 16 14  8  7 15 11 16 14  8]
[ 1  9 13 15 16  8  4  2  1  9 13 15 10  5 11 14  7]
[ 1 10 15 14  4  6  9  5 16  7  2  3  3 13 11  8 12]
[ 1 13 16  4  1 13 16  4  1 13 16  4  1 13 16  4  1]
[ 1  5  8  6 13 14  2 10 16 12  9 11  0  0  0  0  0]
[ 1 15  4  9 16  2 13  8  1 15  4  9 16  2 13  8  1]
[ 1 11  2  5  4 10  8  3 16  6 15 12  3 16  6 15 12]
[ 1 16  1 16  1 16  1 16  1 16  1 16 14  3 14  3 14]
[ 1 14  9  7 13 12 15  6 16  3  8 10  3  8 10  4  5]
[ 1  8 13  2 16  9  4 15  1  8 13  2  0  0  0  0  0]
[ 1  7 15  3  4 11  9 12 16 10  2 14 14 13  6  8  5]
[ 1  4 16 13  1  4 16 13  1  4 16 13  6  7 11 10  6]
[ 1 12  8 11 13  3  2  7 16  5  9  6 12  8 11 13  3]
[ 1  2  4  8 16 15 13  9  1  2  4  8 11  5 10  3  6]
[ 1  6  2 12  4  7  8 14 16 11 15  5 10  9  3  1  6]

ligningssystem på reduceret-række-echelon-form:
[ 1  0  0  0  0  0  0  0  0  0  0  0  0 15  4  9 16]
[ 0  1  0  0  0  0  0  0  0  0  0  0  0 15  2 13  8]
[ 0  0  1  0  0  0  0  0  0  0  0  0  0  2 11 10 14]
[ 0  0  0  1  0  0  0  0  0  0  0  0  0  7  5  5  5]
[ 0  0  0  0  1  0  0  0  0  0  0  0  0  5 14  8 16]
[ 0  0  0  0  0  1  0  0  0  0  0  0  0 14 11  2 15]
[ 0  0  0  0  0  0  1  0  0  0  0  0  0  4  6 10  4]
[ 0  0  0  0  0  0  0  1  0  0  0  0  0 13 12  7  8]
[ 0  0  0  0  0  0  0  0  1  0  0  0  0  3  7  7  0]
[ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  3  7  7]
[ 0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  3  7]
[ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  3]
[ 0  0  0  0  0  0  0  0  0  0  0  0  1 15  4  9 16]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

```
x = PolynomialRing(GF(17),'x').gen()
Q0=1+9*x+3*x^2+12*x^3+1*x^4+2*x^5+13*x^6+9*x^7+0*x^8+10*x^9+10*x^(10)+14*x
    ^(11)
print "Q0=" , Q0
print ""
Q1=1+0*x+0*x^2+0*x^3+1*x^4
print "Q1=" , Q1
print ""
minusf = Q0.quo_rem(Q1)
print "-f =" , minusf
print ""
f=-(minusf[0])
print "f=" , f
print ""
y34=matrix([[f(1),f(3),f(9),f(10),f(13),f(5),f(15),f(11),f(16),f(14),f(8),f
    (7),f(4),f(12),f(2),f(6)]])
print "y34=" , y34
```

Dette giver outputtet

```
Q0= 14*x^11 + 10*x^10 + 10*x^9 + 9*x^7 + 13*x^6 + 2*x^5 + x^4 + 12*x^3 + 3*
    x^2 + 9*x + 1

Q1= x^4 + 1

-f = (14*x^7 + 10*x^6 + 10*x^5 + 12*x^3 + 3*x^2 + 9*x + 1, 0)

f= 3*x^7 + 7*x^6 + 7*x^5 + 5*x^3 + 14*x^2 + 8*x + 16

y34= [ 9 15 10  3  1  0  9  3 14  3  0 14  6 12 11 10]
```

```
#Algoritme 12 punkt 14
e1=y2-y22%17
print "e1=" , e1
print ""
e2=y12-y13%17
print "e2=" , e2
print ""
e3=y33-y34%17
print "e3=" , e3
print ""
c1=y1-e2
print "c1=" ,c1
print ""
c2=y2-e1
print "c2=" ,c2
print ""
c3=y3-e3
print "c3=" ,c3
print ""
c=matrix(GF(17)
    ,[[16,4,14,14,14,11,6,1,16,4,9,1,16,5,3,9,9,12,2,2,14,15,4,3,16,10,13,
2,3,11,14,9,5,13,1,11,13,14,7,11,1,6,5,16,0,11,14,8]])
```

```
print "c =" , c
print ""
mS=G.solve_left(c)
print "mS =" , mS
print ""
Sinv=S^(-1)
mm=mS*Sinv
print "m =" , mm
```

Dette giver outputtet

```
e1= [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  2 15]

e2= [0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0]

e3= [0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0]

c1= [16  4 14 14 14 11  6  1 16  4  9  1 16  5  3  9]

c2= [ 9 12  2  2 14 15  4  3 16 10 13  2  3 11 14  9]

c3= [ 5 13  1 11 13 14  7 11  1  6  5 16  0 11 14  8]

c = [16  4 14 14 14 11  6  1 16  4  9  1 16  5  3  9  9 12  2  2 14 15  4
    3 16 10 13  2  3 11 14  9  5 13  1 11 13 14  7 11  1  6  5 16  0 11 14
     8]

mS = [16  1 15  4  6 12 13  6 13 11  7  7 12  6  2  7 15  6  6 11 14 16 13
     0  8  8  1]

m = [11 15  4 14  9 14  7  0 13  5  4  0  3  1 13  9 12 12  1  0 15  7  0
    9  4  1  0]
```