TEKST NR 187 1990

# RSA

## Et kryptografisk system

af

Annemette Sofie Olufsen
Lars Frellesen
Ole Møller Nielsen

Vejledt af

Michael Pedersen
Finn Munk

TEKSTER fra

IMFUFA ROSKILDE UNIVERSITETSCENTER
INSTITUT FOR STUDIET AF MATEMATIK OG FYSIK SAMT DERES FUNKTIONER I UNDERVISNING, FORSKNING OG ANVENDELSER

IMFUFA, Roskilde Universitetscenter, Postbox 260, 4000 Roskilde

RSA - Et kryptografisk system
af
Annemette Olufsen, Lars Frellesen og Ole Møller Nielsen.
Vejledere: Michael Pedersen og Finn Munk.

IMFUFA tekst nr. 187/90, RUC. 278 sider. ISSN 0106-6242

---

# Abstract

RSA er navnet på et såkaldt offentlig-nøgle kryptosystem, som benyttes mere og mere til beskyttelse af følsomme datatransmissioner.

Et offentlig-nøgle kryptosystem er karakteriseret ved, at det muliggør fortrolig kommunikation uden forudgående udveksling af hemmelige krypteringsnøgler. En konsekvens af dette er blandt andet, at alle transmissionslinier kan være offentlige uden at datasikkerheden af den grund kompromitteres.

Teksten gennemgår den nødvendige talteoretiske baggrund for RSA kryptosystemet, beskriver dets virkemåde og anvendelser samt diskuterer, hvorledes en virkelig implementering kan designes.

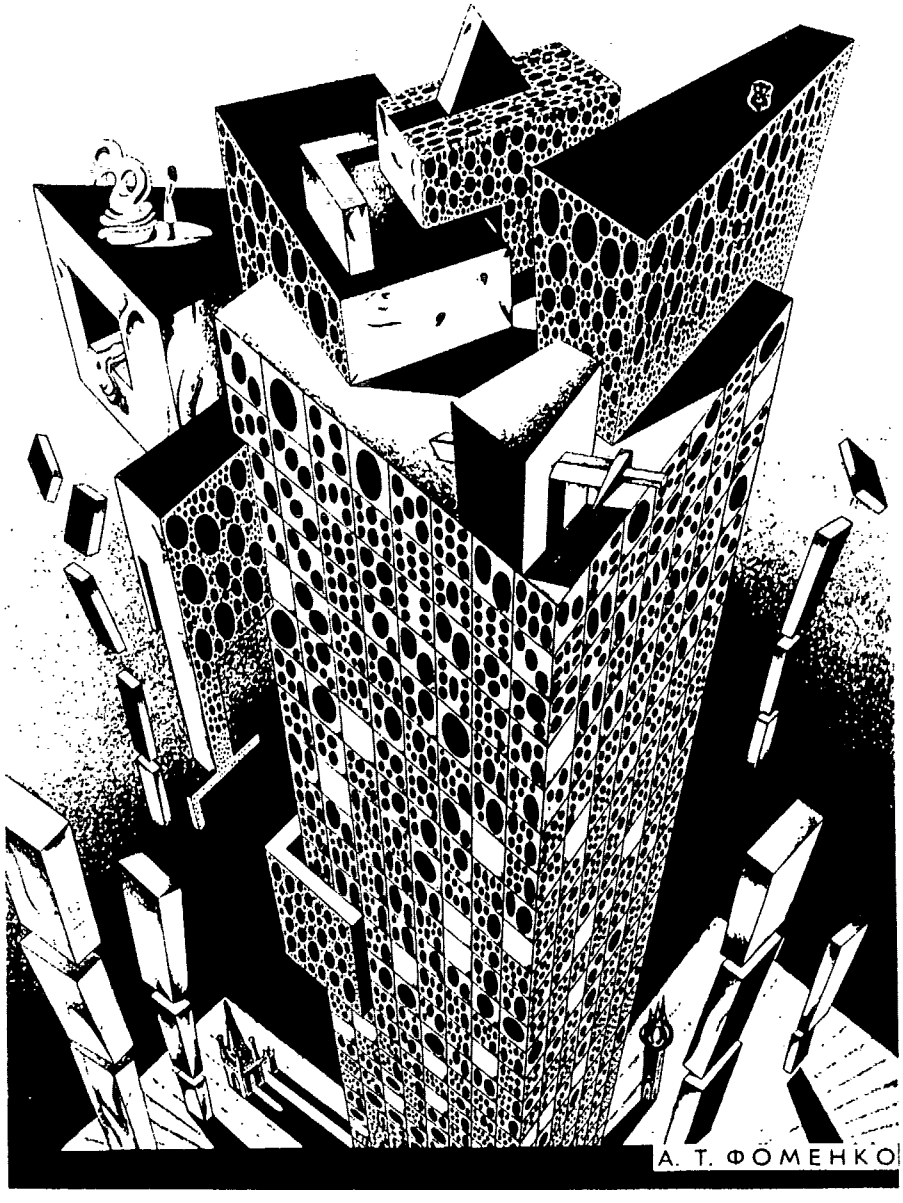
# Forord

Dette projekt handler om RSA-kryptosystemet. Det er lavet i forårs- og efterårssemestrene 1989, og er et kombinationsprojekt mellem matematik (modul 1) og datalogi (modul 2).

I forbindelse med arbejdet på dette projekt, har vi fået hjælp fra forskellige personer, som vi gerne vil sige tak til. Det er Bjørn Hansen fra Pengeinstitutternes BetalingsService (PBS), Peter Landrock fra Århus Universitet, Ronald L. Rivest, hans sekretær Be Hubbard, og Gustavus J. Simmons, samt selvfølgelig vores vejledere Michael Pedersen og Finn Munk.

*Ole Møller Nielsen* (matematik og datalogi)
*Mette Olufsen* (matematik og datalogi)
*Lars Frellesen* (matematik)

RUC, Januar 1990


А. Т. ФОМЕНКО

# Indhold

# Kapitel 1

# Indledning

## 1.1 Indledning

Hver gang man hæver penge i en Dankort betalingsautomat, åbner for fjernsynet eller telefonerer, involverer det *datakommunikation* - dvs kommunikation via et elektronisk medium. Igennem de sidste 30-40 år er forbruget af elektronisk kommunikation eksploderet, og det har åbnet for en ny slags kriminalitet - tyveri eller ændring af fortrolige transmissioner. Det er på mange måder lettere (og mindre risikabelt) at stjæle koden til en bankkonto end at tømme en pansret pengetransport. For at afværge den slags trusler mod datatransmissionen kan man i vid udstrækning benytte kryptologi - *læren om at skjule*. Dette har været kendt i mere end 2000 år indenfor militæret, men anvendelserne er blevet mangedoblet og udviklet kolossalt i det sidste tiår.

I denne rapport fokuserer vi på et bestemt kryptosystem - RSA, der er et såkaldt offentlignøgle kryptosystem. Dette betyder at systemet kan benyttes uden forudgående *hemmelig* aftale. Systemet bygger på talteoretiske opdagelser, som er flere hundrede år gamle, men som hidtil kun har haft anvendelser indenfor matematikkens verden. Som sådan er emnet et eksempel på at den matematiske teori kom *før* behovet for en praktisk anvendelse.

Da emnet har dybe rødder i talteorien samt praktisk relevans i EDB-sammenhæng, er det naturligt at anlægge en tværfaglig synsvinkel i studiet af det. Derfor har vi sat det centrale emne - kryptologi og datasikkerhed - i højsædet og kun medtaget de elementer fra henholdsvis datalogi og matematik, som har direkte relevans til dette. Netop derfor

skulle rapporten da også kunne vinde interesse hos *ikke* matematiske dataloger og *u*praktiske matematikere.

Da vi ønsker at belyse emnet både matematisk og datalogisk samt inddrager en samfundsmæssig dimension, kan problemformuleringen udformes som følgende tre spørgsmål

1. *Hvilken relevans til datasikkerhed har offentlignøgle kryptosystemet RSA ?*
2. *Hvorledes fungerer RSA, og er det et sikkert system ?*
3. *Hvordan kan RSA implementeres i et realistisk EDB-system.*

Disse spørgsmål udgør den røde tråd i rapporten.

## 1.2 Læsevejledning

Denne rapport er som sagt blevet til som et tværfagligt arbejde i grænseområdet mellem datalogi og matematik. Derfor henvender den sig i sin helhed til læsere, der har interesse for begge felter, og i særdeleshed til dem der opdyrker dette grænseområde. Da dette er en meget snæver målgruppe, har vi bestræbt os på at opbygge rapporten i moduler, der kan læses mere eller mindre uafhængigt af resten og, som hver for sig henvender til et lidt bredere publikum. Således kan den matematisk orienterede læser nøjes med at læse rapportens første to dele, da den sidste del beskriver den praktiske implementering af et RSA-kryptosystem. Omvendt vil en datalogisk orienteret læser kunne springe rapportens første del over, men må så *tro* på de matematisk baserede diskussioner i resten af rapporten.

Vi har skrevet rapporten, så vi selv tror vi kunne have forstået den inden vi gik igang med projektet.

Herunder resumerer vi indholdet, så læseren kan danne sig et overblik

**DEL I** DET MATEMATISKE GRUNDLAG.

Her gennemgås den matematik, der er relevant for kryptologien. Denne del kan enten læses uafhængigt af resten, eller springes over, hvis læseren i første omgang ønsker at gøre sig bekendt med

det mere centrale emne i Del II. Endelig kan denne del tjene som opslagsmateriale efterhånden, som der opstår spørgsmål.

Dette fordeles på 3 kapitler :

**Kapitel 2. Grundlæggende talteori**
Dette kapitel beskriver de grundlæggende talteoretiske begreber som vi bruger i resten af rapporten.

**Kapitel 3. Kompleksitetsteori**
Kompleksitetsteorien er primært medtaget af hensyn til projektets datalogiske indhold, men er placeret her på grund af dets matematiske og forudsætningsgivende karakter.

**Kapitel 4. Primtalstest og faktoriseringsalgoritmer**
I dette kapitel gennemgås og bevises algoritmer til at teste om et tal er et primtal og algoritmer til at opløse et sammensat tal i dets primfaktorer. Resultaterne herfra danner grundlaget for RSA-kryptosystemets sikkerhed.

**DEL II** KRYPTOLOGI
Denne del er så at sige "hjertet" i rapporten. Her beskrives hvilken rolle kryptologi spiller indenfor datasikkerhed, samt specielt hvordan offentlignøgle-systemet RSA lader sig benytte til kryptologiske formål. Kryptologidelen består af følgende kapitler:

**Kapitel 5. EDB-sikkerhed**
Emnet EDB-sikkerhed rummer mange af de problemer, der kan afhjælpes ved brug af kryptografi.

**Kapitel 6. Kryptologi**
Her defineres begreberne og vi gennemgår forskellige kryptografiske metoder.

**Kapitel 7. Offentlignøgle kryptografi**
Offentlignøgle kryptografi har en mængde forskellige anvendelser, som vi præsenterer her.

**Kapitel 8. RSA-kryptosystemet**
Det mest kendte offentlig nøgle system hedder RSA. Vi gennemgår systemets virkemåde i dets grundform.

**Kapitel 9. RSA-sikkerhed**
Vi diskuterer en række angreb på RSA-systemet, samt hvordan systemet kan forbedres til at imødekomme disse angreb.

**DEL III** IMPLEMENTERING

I denne del diskuterer vi mulighederne for at implementere RSA-systemet i forskellige EDB-sammenhænge.

**Kapitel 10. Design af et RSA-kryptosystem**

Hvilke overvejelser vi har gjort os med henblik på at implementere RSA.

**Kapitel 11. Implementering**

Her gennemgår vi hvordan de mest interessante algoritmer (f.eks primtalstest, restklassearitmetik) er blevet implementeret. Vi diskuterer også strategien for test af applikationen.

**Kapitel 12. Anvendelse af RSA**

Her beskriver vi dels hvordan RSA-systemet kan bruges i forbindelse med elektronisk post, og dels hvordan det faktisk er realiseret i Dankort netværket.

**Kapitel 13. Konklusion**

Opsummering på rapporten, og besvarelse af problemformuleringen.

Endelig er der en procesbeskrivelse, en symbolliste, en litteraturliste og et index. Til litteraturlisten har vi tilføjet en kort "kildekritik", som afspejler kildens formidlingsmæssige kvalitet - noget som utvivlsomt kan være til gavn for enhver, der ønsker at gå videre med dette emne.

Hvor intet andet angives foregår alle beregninger med hele tal $\mathbb{Z}$.

# Del I

# Det matematiske grundlag

# Kapitel 2

# Grundlæggende Talteori

Dette kapitel giver den talteoretiske baggrund, der er nødvendig for resten af projektet. Kapitlet bygger på kilderne [1,16,31,34,50].

## 2.1 Gruppeteori

I dette afsnit vil vi beskrive nogle af de grundlæggende begreber indenfor gruppeteori.

### 2.1.1 Kompositioner

En afbildning $*$ af $E \times E$ ind i $E$, hvor $E$ er en vilkårlig mængde, kaldes en **komposition**. Hvis $E$ er en talmængde, kan det f.eks. være addition eller multiplikation. En mængde forsynet med en eller flere kompositioner kaldes en **algebraisk struktur**, hvilket skrives $(E, *)$.

Kompositionen $*$ kaldes **associativ**, hvis den associative lov gælder :

$$\forall x, y, z \in E \; : \quad (x * y) * z = x * (y * z)$$

En algebraisk struktur $(E, *)$ hvor $*$ er associativ, kaldes en **semigruppe**. Kompositionen kaldes **kommutativ**, hvis der for alle $x, y$ tilhørende $E$ gælder at :

$$\forall x, y \in E \; : \quad x * y = y * x$$

En semigruppe $(E, *)$, hvor $*$ er kommutativ, kaldes en **kommutativ semigruppe**.

Lad $(E, *)$ være en algebraisk struktur. En ikke tom delmængde $H$ af $E$ kaldes **stabil** overfor kompositionen $*$ hvis :

$$\forall x, y \in H \ : \quad (x * y) \in H$$

Vi benytter skrivemåden $a^{n*}$ for $\overbrace{a * a * \ldots * a}^{n}$, F.eks. er $a^{3*} = a * a * a$. Man kan se, at de sædvanlige potensregneregler gælder i en kommutativ semigruppe. F.eks. er

$$a^{2*} * a^{3*} = a^{5*}$$

### 2.1.2 Neutralt og inverst element

Et element $e$ i en algebraisk struktur $(E, *)$ kaldes **neutralt**, hvis der for ethvert element $x$ tilhørende $E$ gælder at :

$$x * e = e * x = x$$

**Sætning 2.1** *Der findes højst et neutralt element i en algebraisk struktur.*

**Bevis :**

Hvis $e$ og $f$ begge er neutrale elementer i $(E, *)$, har vi både $e * f = e$ fordi $e$ er neutralt, og $e * f = f$ fordi $f$ er neutralt. Altså er $e = f$. □

Et element $a$ i en kommutativ semigruppe $(E, *)$ med et neutralt element $e$, har et **inverst element**, hvis ligningssystemet 2.1 har en løsning.

$$x * a = e \quad \text{og} \quad a * x = e \tag{2.1}$$

Det inverse element $x$ skrives $a^{-1*}$. Hvis $a$ har et inverst element, er $a$ **invertibel**.

### 2.1.3 Grupper

**Definition 2.1 (Gruppe)**
*En algebraisk struktur $(G, *)$ kaldes en* **gruppe**, *hvis den har følgende egenskaber :*

1. $*$ *er associativ.*
2. *Der findes et neutralt element e i G.*
3. *Ethvert element i G har et inverst element, som også ligger i G.*

Hvis kompositionen $*$ yderligere er kommutativ, kaldes det for en **kommutativ** eller **Abelsk gruppe**. Et eksempel på en kommutativ gruppe er mængden af hele tal $\mathbb{Z}$ med kompositionen addition : $(\mathbb{Z}, +)$.

Lad $(G, *)$ være en gruppe. Hvis $H$ er en stabil delmængde af $G$ med hensyn til kompositionen $*$ og $(H, *)$ selv er en gruppe, kaldes $(H, *)$ en **undergruppe** til $(G, *)$.

### 2.1.4 Ringe og legemer

Hvor grupper kun er defineret for een kompositionsregel, kan man med ringe og legemer beskrive *samspillet* mellem *to* kompositioner på en mængde.

**Definition 2.2 (Ring)**
*En ring er en algebraisk struktur $(R, *, \diamond)$ med mindst to elementer, hvor kompositionsreglerne $*$ og $\diamond$ er defineret overalt i R, hvis der gælder følgende :*

1. $(R, *)$ *er en kommutativ gruppe.*
2. $(R, \diamond)$ *er en* semi*gruppe.*
3. $\diamond$ *er distributiv over* $*$*, hvilket vil sige at :*

$$\forall a, b, c \in R : \quad a \diamond (b * c) = (a \diamond b) * (a \diamond c)$$

Hvis $(R, \diamond)$ ydermere er en kommutativ semigruppe taler vi om en **kommutativ ring**. Et eksempel på en kommutativ ring er $(\mathbb{Z}, +, \cdot)$, idet $(\mathbb{Z}, +)$ er en gruppe og $(\mathbb{Z}, \cdot)$ er en semigruppe.

**Definition 2.3 (Legeme)**
*En ring $(L, *, \diamond)$ kaldes et legeme, hvis det ud over egenskaberne fra definition 2.2 gælder at $(L \setminus \{e\}, \diamond)$ er en gruppe.*

Et eksempel på et legeme er $(\mathbb{R}, +, \cdot)$, idet både $(\mathbb{R} \setminus \{0\}, \cdot)$ og $(\mathbb{R}, +)$ er grupper.

### 2.1.5 Endelige grupper

Hvis en gruppe $(G, *)$ har et endeligt antal elementer, kaldes gruppen for endelig. Antallet af elementer i gruppen kaldes for **gruppens orden** : $\mathrm{ORD}(G)$.

**Sætning 2.2** *Når $(H, *)$ er en undergruppe til en endelig gruppe $(G, *)$, gælder der at*

$$\mathrm{ORD}(G) = k \cdot \mathrm{ORD}(H) \text{ hvor } k \in \mathbb{N}.$$

Beviset findes i [1].

**Sætning 2.3** *Lad $G$ være en endelig gruppe. For ethvert element $a$ i $G$, findes der et $p$ så :*

$$a^{p*} = e$$

**Bevis :**

Lad $a$ være et vilkårligt element i $(G, *)$ og kald $\mathrm{ORD}(G)$ for $n$. Elementerne

$$a^{1*}, a^{2*}, \ldots, a^{(n+1)*}$$

tilhører alle $G$, men da deres antal $(n+1)$ overstiger antallet af elementer i $G$, må der mindst være to, der er ens :

$$a^{s*} = a^{r*} \quad \text{og} \quad r < s$$

Heraf fås

$$\begin{aligned} a^{s*} &= a^{r*} \quad \Leftrightarrow \\ a^{s*} * a^{-r*} &= e \quad \Leftrightarrow \\ a^{(s-r)*} &= e \end{aligned}$$

Vi kan så sætte $p = s - r$ så $a^{p*} = e$. □

For ethvert $a$ findes altså en potens af $a$, der er lig med det neutrale element. Det mindste positive heltal $p$, der opfylder dette kaldes **elementets orden** i $G$ og betegnes $\mathrm{ORD}(a, G)$.

**Sætning 2.4** *hvis $G$ er en endelig gruppe, vil potenserne af et element $a$ i $G$, op til dets orden $p$ :*

$$a^{1*}, a^{2*}, \ldots, a^{p*}$$

*alle være forskellige, og udgør sammen med kompositionen $*$ en undergruppe til $(G, *)$.*

**Bevis :**

Vi antager at $1 \leq r < s \leq p$, og at to af eksponenterne er ens - altså at en modstrid til sætningen gælder.

$$\begin{aligned} a^{s*} &= a^{r*} \quad \Leftrightarrow \\ a^{(s-r)*} &= e \end{aligned}$$

Da $0 < (s-r) < p$ og $p$ er den mindste eksponent, hvor $a^{p*} = e$ har vi en modstrid, hvorfor alle potenser op til elementets orden er forskellige.

At mængden

$$A = \{a^{1*}, a^{2*}, \ldots, a^{p*}\}$$

er en undergruppe til $(G, *)$, kan ses således :

$$a^{r*} * a^{s*} = a^{(r+s)*}$$

Hvis $r+s < p$ har vi umiddelbart at $a^{r*} * a^{s*} \in A$ og hvis $r+s \geq p$ får vi at

$$\begin{aligned} a^{(r+s)*} &= a^{(r+s-p)*} * a^{p*} \\ &= a^{(r+s-p)*} * e \\ &= a^{(r+s-p)*} \end{aligned}$$

og da

$$r + s < 2p \text{ er } r + s - p < p$$

og vi får at :

$$\forall r, s : \quad a^{(r+s)*} \in A$$

$A$ er altså stabil med hensyn til $*$. $A$ indeholder det neutrale element idet $a^{p*} = e$, og alle elementer er invertible da :

$$a^{s*} * a^{(p-s)*} = a^{p*} = e$$

□

Da $A$ er en undergruppe, har vi ifølge sætning 2.2 at :

$$\mathrm{ORD}(G) = k \cdot \mathrm{ORD}(a, G) \text{ hvor } k \in \mathbb{N}$$

**Definition 2.4 (Frembringer)**
*Et element a kaldes en* **frembringer**, *hvis dets orden er lig gruppens orden :*

$$\mathrm{ORD}(a, G) = \mathrm{ORD}(G)$$

*Potenserne af a, vil gennemløbe eller* frembringe *alle elementer i gruppen G.*

## 2.2 Delelighed og primtal

### 2.2.1 Delelighed

**Definition 2.5** *Tallet a er divisor i b, hvis der findes et d således at* $b = d \cdot a$. *Dette udtrykkes med notationen*

$$a|b$$

Ethvert tal numerisk større end 1, har mindst to positive divisorer, 1 og tallet selv. En divisor forskellig fra disse to, kaldes en **ikke-triviel divisor**. Har tallet kun de to **trivielle divisorer** kaldes det et **primtal**. Alle andre hele tal kaldes **sammensatte**.

Tallet 0 har uendelig mange divisorer, idet $a|0$ for alle $a \in \mathbb{Z}$. Alle andre tal $b \in \mathbb{Z}$ har kun et endeligt antal divisorer, da

$$a|b \Rightarrow -b \leq a \leq b$$

og da intervallet $[-b; b]$ indeholder *endeligt* mange hele tal.

Det gælder for ethvert heltal $b$, at det kan skrives på formen :

$$b = qa + r, \quad q, a, r \in \mathbb{Z}$$

Man siger at

$b$ divideret med $a$ er lig $q$ med $r$ til rest.

Resten $r$ er altså det der bliver til overs ved heltalsdivision af $b$ med $a$. Heltalsdivision symboliseres ved $b//a = q$. Hvis $r$ ligger i intervallet $[0, b[$, kaldes $r$ **den principale rest.**

Ud fra den principale rest, kan man definere **Modulus operatoren** :

$$b \text{ MOD } a = r \text{ ( MOD læses "modulo").}$$

Denne funktion må ikke forveksles med den gængse matematiske modulo, som beskrives i afsnit 2.4.1.

Der gælder specielt at

$$b \text{ MOD } a = 0 \Leftrightarrow a|b$$

### 2.2.2 Aritmetikkens fundamentalsætning

Aritmetikkens fundamentalsætning siger, at et tal på kun en måde kan skrives som et produkt af primtal. Inden vi viser dette, skal vi dog bruge et par sætninger.

**Sætning 2.5** *Ethvert heltal $n > 1$ er enten et primtal eller et produkt af primtal.*

**Bevis :**

Vi beviser det ved induktion.

**Induktionsstart:** For $n = 2$ er sætningen umiddelbart sand.

**Induktionstrin :** Antag så at sætningen er sand for alle tal op til $n - 1$. Hvis $n$ er et primtal, er sætningen sand. Hvis $n$ er sammensat, har den en divisor $d$, således at

$$n = c \cdot d.$$

Da $d$ er en *ikke-triviel* divisor, må det gælde at

$$d, c \in ]1, n[$$

Ifølge induktionshypotesen er $d$ og $c$ enten primtal, eller produkter af primtal. Derfor må $n$ også være et produkt af primtal.

□

**Sætning 2.6** *Hvis et primtal $p$ går op i $ab$, går det enten op i $a$ eller $b$. Mere generelt gælder det at såfremt $p$ går op i $a_1 \cdot a_2 \cdots a_n$, går det op i mindst en af faktorerne.*

**Bevis :**

Antag at

$$p|ab \text{ og } p \not| a$$

Da $a$ kan skrives som $a = mp + r$ fås

$$\begin{aligned} p \mid ab &\Leftrightarrow \\ ab &= kp \Leftrightarrow \\ mpb + rb &= kp \Leftrightarrow \\ rb &= p(k - mb) \end{aligned}$$

og da $p \not| r$ gælder at $p|b$. □

**Sætning 2.7 (Aritmetikkens fundamentalsætning)** *Et tal $n$ kan på en og kun en måde (bortset fra rækkefølgen) skrives som et produkt af primtal :*

$$n = p_1 \cdot p_2 \cdot \ldots \cdot p_s$$

**Bevis :**

Vi beviser det ved at benytte induktion over $n$.

**Induktionsstart:** For $n = 2$ er sætningen umiddelbart sand.

**Induktionstrin :** Antag at sætningen gælder for alle tal op til $n - 1$. Er $n$ et primtal, er vi færdige. Antag derfor at $n$ er et sammensat tal. Fra sætning 2.5 ved vi, at $n$ *kan* skrives som et produkt af primtal. Vi skal så blot bevise, at der kun er en måde at gøre det på. Antag nu at $n$ på to måder kan skrives som et produkt af primtal :

$$n = p_1 p_2 p_3 \ldots p_s = q_1 q_2 q_3 \ldots q_t \tag{2.2}$$

Vi skal så vise at $s = t$ og at hvert $p$ er lig et $q$.

Da $p_1$ går op i $n$ og derfor også i produktet $q_1q_2q_3 \ldots q_t$, må den ifølge sætning 2.6 gå op i mindst en af faktorerne. Vi kan nu antage, at $q$'erne er skrevet op, så $p_1$ går op i $q_1$. Da $q_1$ og $p_1$ begge er primtal, må $p_1$ være lig med $q_1$, og vi kan derfor forkorte ligning 2.2 med $p_1$, hvorefter vi får :

$$\frac{n}{p_1} = p_2p_3 \ldots p_s = q_2q_3 \ldots q_r$$

Da $p_1 > 1$ og dermed $n/p_1 < n$, fortæller induktionshypotesen os at sætningen gælder for $n/p_1$. Da $n$ på kun en måde kan skrives som

$$n = \frac{n}{p_1} \cdot p_1$$

må sætningen også gælde for $n$.

□

### 2.2.3 Primtalsfordeling

Primtallene dukker op i talrækken på en så tilfældig måde, at man ikke kan sige, hvornår de kommer og hvor store spring der er fra primtal til primtal. Alligevel kan man godt sige noget om *fordelingen* af primtallene.

**Definition 2.6** *Funktionen $\pi(x)$ er lig antallet af primtal i intervallet $[2; x]$.*

$\pi(x)$ kan ikke beregnes præcist, men i følge [1] gælder der følgende :

$$\lim_{x \to \infty} \frac{\pi(x) \log x}{x} = 1$$

Af dette følger, at den gennemsnitlige afstand mellem to primtal, af størrelsesorden $x$ er $\log x$, og at sandsynligheden for, at et tal $x$ er et primtal, er $\frac{1}{\log x}$.

## 2.3 Den største fælles divisor

### 2.3.1 Den største fælles divisor

**Definition 2.7 (Største fælles divisor)**
*Hvis der for tal $u, v, d \in \mathbb{Z}$ gælder at :*

$$d|u \text{ og } d|v$$

*kaldes $d$ en* **fælles divisor** *for $u$ og $v$.*
*Hvis der for alle $t > d$ endvidere gælder at :*

$$t \nmid u \text{ eller } t \nmid v$$

*kaldes $d$ for* **den største fælles divisor** *af $u$ og $v$. Dette skrives*

$$d = \mathrm{SFD}(u, v)$$

*Er enten $u$ eller $v$ lig 0, er defineres $d$ som henholdsvis $v$ eller $u$.*

**Sætning 2.8** *Lad $d = \mathrm{SFD}(u, v)$. Der gælder så, at alle andre fælles divisorer i $u$ og $v$ også er divisorer i $d$.*

**Bevis :**

Fra aritmetikkens fundamentalsætning (sætning 2.7) ved vi at $u$ og $v$ kan opløses i primfaktorer som

$$u = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \ldots p_n^{\alpha_n} \text{ og } v = r_1^{\beta_1} \cdot r_2^{\beta_2} \ldots r_n^{\beta_n}$$

Enhver *fælles* divisor $d' = q_1^{\alpha_1} q_2^{\alpha_2} \ldots q_n^{\alpha_n}$ må derfor være sammensat af primfaktorer, som *både* optræder i $u$ og $v$. Den største fælles divisor må derfor indeholde *alle* de fælles primfaktorer fra $u$ og $v$. Enhver fælles divisor $d'$ må da være divisor i $\mathrm{SFD}(u, v)$. □

Da mængden af primfaktorer i både $u$ og $v$ er endelig, følger det af sætning 2.8 at der eksisterer netop *en* største fælles divisor. Af sætningen følger endvidere, at hvis $u$ og $v$ ikke har nogle fælles primfaktorer, så er

$$d = \mathrm{SFD}(u, v) = 1$$

Man siger da at $u$ og $v$ er **parvist primiske**.

Skal en række tal være **indbyrdes primiske**, skal de to og to være parvist primiske.

**Definition 2.8 (Det mindste fælles multiplum - MFM)**
*Lad* $(u, v) \in \mathbb{Z}^2$. *Så er* MFM$(u, v)$ *lig med det mindste tal* $m$, *hvor det gælder at* $u|m$ *og* $v|m$.

### 2.3.2 Euclids algoritme

For over 2000 år siden fandt Euclid en meget hurtig metode til at beregne SFD$(u, v)$. Algoritmen behøver kun meget få divisioner for at give et resultat, fordi resten halveres ved hver andet skridt[1].

**Sætning 2.9** *Hvis* $d$ *er divisor i både* $u$ *og* $v$, *vil* $d$ *også være divisor i* $(u - v)$ *såfremt* $u > v$. *Mere generelt gælder det, at hvis* $d$ *er divisor i* $u$ *og* $v$, *vil* $d$ *også være divisor i* $u$ MOD $v$.

**Bevis :**

Da vi har forudsat at $d$ er divisor i både $u$, og $v$ vil

$$\frac{u}{d} - \frac{v}{d} = \frac{(u - v)}{d}$$

hvorfor $d$ også er divisor i $u - v$. Anvender vi dette flere gange, får vi at

$$\begin{aligned} d|u \text{ og } d|v &\Rightarrow d|(u - v) \\ &\Rightarrow d|((u - v) - v) \\ &\vdots \\ &\Rightarrow d|(u - nv) \end{aligned}$$

Da $u$ kan skrives som $nv + r$, kan vi vælge et $n$ således at $r$ ligger i intervallet $[0; v[$, hvilket giver at

$$r = u - nv = u \text{ MOD } v$$

Dermed har vi vist at $d|(u \text{ MOD } v)$. □

[1] se afsnit 3.2.2 om algoritmens kompleksitet

På den måde kan man rekursivt finde den største fælles divisor, som følger

$$\begin{aligned} \mathrm{SFD}(u,v) &= \mathrm{SFD}(v,(u \text{ MOD } v)) \\ &\text{og} \\ \mathrm{SFD}(u,0) &= u \end{aligned}$$

Lad $r_i$ betyde resten ved den $i$'te division. Gennemløbes algoritmen $n$ gange fås forløbet :

$$\begin{aligned} \mathrm{SFD}(u,v) = \mathrm{SFD}(v,r_1) &= \mathrm{SFD}(r_1,r_2) = \ldots \\ &= \mathrm{SFD}(r_{n-1},r_n) = \\ &= \mathrm{SFD}(r_n,0) = r_n \end{aligned} \tag{2.3}$$

Af nedenstående eksempel kan man se, at denne algoritme bruger *meget* få gennemløb (ca et pr decimalt ciffer i $u$), til at finde den største fælles divisor.

***Eksempel 2.1. Euclids algoritme***

$$\begin{aligned} \mathrm{SFD}(182.135.106, 13.974.858) &= \\ \mathrm{SFD}(13.974.858, 461.952) &= \\ \mathrm{SFD}(461.952, 116.298) &= \\ \mathrm{SFD}(116.298, 113.058) &= \\ \mathrm{SFD}(113.058, 3.240) &= \\ \mathrm{SFD}(3.240, 2.898) &= \\ \mathrm{SFD}(2.898, 342) &= \\ \mathrm{SFD}(342, 162) &= \\ \mathrm{SFD}(162, 18) &= \\ \mathrm{SFD}(18, 0) &= 18 \end{aligned}$$

**Sætning 2.10** *Den største fælles divisor $d$ af to tal $u$ og $v$ kan udtrykkes som en heltallig linearkombination af dem. Det betyder, at der eksisterer to heltal $x$ og $y$ så*

$$d = u \cdot x + v \cdot y \tag{2.4}$$

**Bevis :**

Idet vi benytter betegnelserne fra ligning 2.3 og kalder kvotienterne fra de forskellige divisioner for $q_1, q_2, \ldots, q_n$, har vi :

$$\begin{aligned} r_1 &= u - v \cdot q_1 \\ r_2 &= v - r_1 \cdot q_2 \\ r_3 &= r_1 - r_2 \cdot q_3 \\ &\vdots \\ r_n &= r_{n-2} - r_{n-1} \cdot q_n = \mathrm{SFD}(u, v) \end{aligned}$$

hvor resten $r_n$ er $r_{n-2}$ MOD $r_{n-1}$. På tilsvarende måde er :

$$r_{n-1} = r_{n-3} - r_{n-2} \cdot q_{n-1}.$$

Ved at indsætte dette i udtrykket for $r_n$, får man :

$$\begin{aligned} r_n &= r_{n-2} - (r_{n-3} - r_{n-2} \cdot q_{n-1}) q_n \\ &= r_{n-2} + r_{n-2} \cdot q_{n-1} \cdot q_n - r_{n-3} \cdot q_n \\ &= (1 + q_{n-1} \cdot q_n) r_{n-2} - q_n \cdot r_{n-3} \\ &= h_1 \cdot r_{n-3} + k_1 \cdot r_{n-2} \end{aligned}$$

hvor $h_1$ og $k_1$ er hele tal. Ved at fortsætte på denne måde "baglæns" gennem udregningerne i Euclids algoritme, får vi :

$$\begin{aligned} \mathrm{SFD}(u, v) = r_n &= r_{n-2} + k_0 \cdot r_{n-1} \\ &= h_1 \cdot r_{n-3} + k_1 \cdot r_{n-2} \\ &= h_2 \cdot r_{n-4} + k_2 \cdot r_{n-3} \end{aligned}$$

$$\begin{aligned} &= \vdots \\ &= h_{n-2} \cdot r_1 + k_{n-2} \cdot r_2 \\ &= h_{n-1} \cdot v + k_{n-1} \cdot r_1 \\ &= h_n \cdot u + k_n \cdot v \\ &= x \cdot u + y \cdot v \end{aligned}$$

hvor $h_1, k_1, h_2, k_2, \ldots, x$ og $y$ er hele tal. Vi har herved beregnet det ønskede udtryk for SFD$(u, v)$.

Denne algoritme kaldes ofte **Euclids modificerede algoritme**
□

### 2.3.3 Eulers $\phi$-funktion

**Definition 2.9 (Eulers $\phi$-funktion)**
*For ethvert naturligt tal n angiver Eulers $\phi$-funktion antallet af naturlige tal i intervallet* $[1; n]$, *som er* indbyrdes primiske med n.

$$\phi(n) = \#(\{m \in \mathbb{N}, m \leq n | \text{SFD}(m, n) = 1\})$$

Her er en tabel over $\phi(n)$ for $n \in [1, 10]$

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\phi(n)$ | 1 | 1 | 2 | 2 | 4 | 2 | 6 | 4 | 6 | 4 |

Af definitionen kan man se, at hvis $p$ er et primtal er $\phi(p) = p - 1$. Videre kan man se at antallet af tal $m$, der ikke er indbyrdes primiske med $n$, dvs. hvor SFD$(n, m) > 1$, er $n - \phi(n)$.

**Sætning 2.11** *Hvis n er produktet af to forskellige primfaktorer p og q er*

$$\phi(n) = \phi(p) \cdot \phi(q) = (p - 1)(q - 1)$$

**Bevis :**

Da $n$ har divisorerne $p$ og $q$ gælder der for alle tal $m$ af formen

$$\{q, 2q, \ldots, (p-1)q, pq\} \cup \{p, 2p, \ldots, (q-1)p, pq\}$$

at $\mathrm{SFD}(m, n) > 1$.

Antallet af $m$ hvor dette er opfyldt er $p$ fra den første mængde og $q$ fra den anden fratrukket 1, fordi $pq$ optræder i begge mængder, i alt $p + q - 1$. Da der er $pq$ tal i alt, vil antallet af indbyrdes primiske elementer være

$$\begin{aligned} \phi(n) &= pq - (p + q - 1) \\ &= pq - p - q + 1 \\ &= (p-1)(q-1) \\ &= \phi(p) \cdot \phi(q). \end{aligned}$$

□

Sætningen gælder også mere generelt. Er

$$n = p_1 \cdot p_2 \cdot \ldots \cdot p_s$$

fås

$$\phi(n) = \phi(p_1) \cdot \phi(p_2) \ldots \phi(p_s) = (p_1 - 1)(p_2 - 1) \ldots (p_s - 1)$$

***Eksempel 2.2. Eulers $\phi$-funktion***

*Lad $p = 5$ og $q = 2$ så $n = pq = 10$. Sætning 2.11 siger at*

$$\phi(10) = (p-1) \cdot (q-1) = 4 \cdot 1 = 4$$

*Man finder da også at 1, 3, 7 og 9 er primiske med 10.*

Eulers $\phi$-funktion kaldes undertiden **Eulers totient funktion.**

## 2.4 Restklasser og kongruens

Restklasser er et begreb, med hvilket man kan opdele naturlige tal i grupper svarende til, hvilken rest de giver ved division med et positivt heltal $d$.

### 2.4.1 Kongruens

I stedet for at skrive $u$ MOD $m = v$ MOD $m$, har man indført en notation kaldet **kongruens** :

$$u \equiv v \bmod m$$

Det læses *"u er kongruent med v modulo m"*, og betyder at $u$ og $v$ giver samme rest ved division med $m$, så $m|(u - v)$. Hvis specielt $m|a$, får man :

$$a \equiv 0 \bmod m$$

Eksempelvis er :

$$\begin{aligned} 10 &\equiv 0 \bmod 5 \\ 7 &\equiv 2 \bmod 5 \\ 22 &\equiv 2 \bmod 5 \end{aligned}$$

Ser man på kongruensen

$$x \equiv a \bmod n \tag{2.5}$$

hvor $a$ og $n$ er faste tal, vil der være en hel *klasse* af tal, der opfylder sætning 2.5, nemlig alle $x = a + kn$, hvor $k \in \mathbb{Z}$.

### 2.4.2 Restklasser

**Definition 2.10** *To tal $u, v \in \mathbb{Z}$ tilhører samme* **restklasse** *modulo $n$ hvis*

$$u \equiv v \bmod n$$

Ethvert tal $u$ kan henføres til netop en af følgende mængder :

- mængden af tal af formen $kn$
- mængden af tal af formen $kn + 1$
- mængden af tal af formen $kn + 2$
- $\vdots$

- mængden af tal af formen $kn + (n-1)$

Hver mængde udgør en restklasse modulo $n$. Tilsammen kaldes mængderne for *restklasserne* modulo $n$.

Hvis $u$ er et vilkårligt helt tal og $n$ er et positivt helt tal, ser man ofte betegnelsen $(u)_n$ for den restklasse modulo $n$, hvortil $u$ hører. Vi kalder $u$ en **repræsentant** for denne restklasse. Hvis $0 \leq u < n$ kaldes $u$ for den **principale repræsentant** for restklassen $(u)_n$ eller **den principale rest** modulo $n$. Bemærk forskellen mellem MOD og mod . MOD giver den principale rest, hvor mod giver hele restklassen.

### 2.4.3 Restklassegrupper

Mængden af alle restklasser modulo $m$, $m \in \mathbb{Z}$ betegnes $Z_m$ og kaldes ofte den *komplette* mængde af restklasser modulo $m$. Denne mængde er en gruppe med hensyn til addition modulo m.

**Sætning 2.12** *$(Z_m, +)$ er en gruppe.*

**Bevis :**

For at $(Z_m, +)$ kan være en gruppe, skal reglerne på side 9 være opfyldt :

1. *Kompositionen skal være associativ overfor addition.*
   Dette følger af de grundlæggende regneregler :
   Vi kan skrive

$$\begin{aligned} a_1 &= k_1 n + r_1 \\ a_2 &= k_2 n + r_2 \end{aligned}$$

hvor $r_1, r_2 \in [0, n-1]$. Vi har så

$$\begin{aligned} (a_1 + a_2) \text{ MOD } n &= ((k_1 n + r_1) + (k_2 n + r_2)) \text{ MOD } n \\ &= ((k_1 + k_2)n + (r_1 + r_2)) \text{ MOD } n \\ &= (r_1 + r_2) \text{ MOD } n \\ &= ((a_1 \text{ MOD } n) + \\ &\quad (a_2 \text{ MOD } n)) \text{ MOD } n. \end{aligned}$$

| + | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 2 | 3 | 4 | 5 | 0 |
| 2 | 2 | 3 | 4 | 5 | 0 | 1 |
| 3 | 3 | 4 | 5 | 0 | 1 | 2 |
| 4 | 4 | 5 | 0 | 1 | 2 | 3 |
| 5 | 5 | 0 | 1 | 2 | 3 | 4 |

**Tabel 2.1:** Addition i restklasserne modulo 6

2. *Der skal findes et neutralt element i* $(Z_m, +)$.
   Som ved sædvanlig addition, er 0 det neutrale element, hvad man hurtigt kan overbevise sig om.
3. *Ethvert element i* $(Z_m, +)$ *skal være invertibelt.*
   Et element $a$ har det inverse element $(m - a)$, da

$$a + (m - a) \equiv 0 \bmod m$$

□

Regning med restklasser kan illustreres ved skemaer, som det er vist på tabel 2.1. Her er addition vist indenfor restklasserne (modulo 6).

Af figur 2.1 ses det at 1 og 5 er frembringere i $(Z_6, +)$ :

$$1^{1+} = 1 \text{ MOD } 6 = 1$$
$$1^{2+} = 1 + 1 \text{ MOD } 6 = 2$$
$$1^{3+} = 1 + 1 + 1 \text{ MOD } 6 = 3$$
$$1^{4+} = 1 + 1 + 1 + 1 \text{ MOD } 6 = 4$$
$$1^{5+} = 1 + 1 + 1 + 1 + 1 \text{ MOD } 6 = 5$$
$$1^{6+} = 1 + 1 + 1 + 1 + 1 + 1 \text{ MOD } 6 = 0$$

$$5^{1+} = 5 \text{ MOD } 6 = 5$$
$$5^{2+} = 5 + 5 \text{ MOD } 6 = 4$$
$$5^{3+} = 5 + 5 + 5 \text{ MOD } 6 = 3$$

$$
\begin{aligned}
5^{4+} &= 5+5+5+5 \text{ MOD } 6 = 2 \\
5^{5+} &= 5+5+5+5+5 \text{ MOD } 6 = 1 \\
5^{6+} &= 5+5+5+5+5+5 \text{ MOD } 6 = 0
\end{aligned}
$$

**Sætning 2.13** *Lad $G_m$ være mængden af indbyrdes primiske restklasser modulo m, dvs $G_m \subseteq Z_m$. Så er $(G_m, \cdot)$ en gruppe. Hvis m er et primtal er $G_m = Z_m$.*

**Bevis :**

Beviset følger den samme plan som før :

1. *Kompositionen skal være associativ overfor multiplikation.*
   Igen følger dette af de grundlæggende regneregler :

$$
\begin{aligned}
a_1 &= k_1 n + r_1 \\
a_2 &= k_2 n + r_2
\end{aligned}
$$

   hvor $r_1, r_2 \in [0, n-1]$. Vi har her at

$$
\begin{aligned}
(a_1 \cdot a_2) \text{ MOD } n &= ((k_1 n + r_1) \cdot (k_2 n + r_2)) \text{ MOD } n \\
&= ((k_1 k_2 n + r_1 k_2 + r_2 k_1) n + \\
&\quad r_1 r_2) \text{ MOD } n \\
&= (r_1 \cdot r_2) \text{ MOD } n \\
&= ((a_1 \text{ MOD } n) \cdot \\
&\quad (a_2 \text{ MOD } n)) \text{ MOD } n
\end{aligned}
$$

2. *Der skal findes et neutralt element i $(G_m, \cdot)$.*
   Som sædvanlig ved multiplikation er 1 det neutrale element.
3. *Ethvert element i $(G_m, \cdot)$ skal være invertibelt.*
   Dette betyder at ligningen

$$a \cdot x \equiv 1 \bmod m$$

   skal have en løsning. Da vi ved at $\text{SFD}(a, m) = 1$ og dermed at $ax + my = 1$ kan dette $x$ ifølge sætning 2.10 findes.

| · | 1 | 3 | 7 | 9 |
|---|---|---|---|---|
| 1 | 1 | 3 | 7 | 9 |
| 3 | 3 | 9 | 1 | 7 |
| 7 | 7 | 1 | 9 | 3 |
| 9 | 9 | 7 | 3 | 1 |

**Tabel 2.2:** Restklassegruppen modulo 10

Hvis $m$ er et primtal, er alle elementer i mængden indbyrdes primiske, og punkt tre gælder overalt i $Z_m$, hvorfor $G_m = Z_m$. □

Ifølge definition 2.2 og 2.3 kaldes $(Z_m, +, \cdot)$ derfor en **restklasse***ring* modulo $m$, hvis $m$ er sammensat og et **restklasse***legeme*, hvis $m$ er et primtal.

Punkt 3 i beviset, viser at det altid er muligt at finde den inverse til et element, og anviser direkte en metode til at gøre det :

**Sætning 2.14 (Beregning af inverse)**
*Hvis* SFD$(a, n)$ *er 1, har ligningen*

$$a \cdot x \equiv 1 \bmod n \tag{2.6}$$

*en løsning, som kan beregnes med Euclids algoritme.*

Restklasselegemer kaldes også for **Galois legemer**[2] : **GF(p)**, hvor $p$ er et primtal.

Da restklassegruppen $(G_n, \cdot)$ kun indeholder de elementer i $Z_n$, der er indbyrdes primiske med $n$, er der i alt $\phi(n)$ elementer i den. Vi har derfor ORD$(G_n) = \phi(n)$. F.eks vil restklassegruppen modulo 10 have orden 4, da der er fire elementer $\{1, 3, 7, 9\}$, der er indbyrdes primiske med 10. Restklassegruppen modulo 10 vil derfor se ud som i tabel 2.2.

I denne gruppe er 3 og 7 frembringere :

$$3^1 = 3 \text{ MOD } 10 = 3$$

$$3^2 = 3 \cdot 3 \text{ MOD } 10 = 9$$

[2]Et legeme hedder på engelsk : Field.

$$\begin{aligned} 3^3 &= 3 \cdot 3 \cdot 3 \text{ MOD } 10 = 7 \\ 3^4 &= 3 \cdot 3 \cdot 3 \cdot 3 \text{ MOD } 10 = 1 \end{aligned}$$

$$\begin{aligned} 7^1 &= 7 \text{ MOD } 10 = 7 \\ 7^2 &= 7 \cdot 7 \text{ MOD } 10 = 9 \\ 7^3 &= 7 \cdot 7 \cdot 7 \text{ MOD } 10 = 3 \\ 7^4 &= 7 \cdot 7 \cdot 7 \cdot 7 \text{ MOD } 10 = 1 \end{aligned}$$

### 2.4.4 Regneregler for restklasser

**Sætning 2.15** *Lad* $(u, v, d) \in \mathbb{Z}^3$. *Der gælder så følgende :*

$$u \equiv v \bmod d \Leftrightarrow (u - v) \equiv 0 \bmod d$$

**Bevis :**

Sætningen følger af definitionen på kongruensnotationen :

$$\begin{aligned} u &\equiv v \bmod d \Leftrightarrow \\ d &\mid (u - v) \Leftrightarrow \\ (u - v) &\equiv 0 \bmod d \end{aligned}$$

□

**Sætning 2.16** *For et vilkårligt positivt helt tal* $d$ *og fire vilkårlige hele tal* $u, u', v$ *og* $v'$ *gælder at :*

$$\left.\begin{array}{lll} u \equiv u' \bmod d & og \\ v \equiv v' \bmod d \end{array}\right\} \Rightarrow u + v \equiv (u' + v') \bmod d$$

*og*

$$\left.\begin{array}{lll} u \equiv u' \bmod d & og \\ v \equiv v' \bmod d \end{array}\right\} \Rightarrow u \cdot v \equiv (u' \cdot v') \bmod d$$

**Bevis :**

Da $d|(u-u')$ og $d|(v-v')$ har vi at

$$\begin{aligned} d &\mid ((u-u')+(v-v')) \Rightarrow \\ d &\mid ((u+v)-(u'+v')) \Rightarrow \\ ((u+v)-(u'+v')) &\equiv 0 \bmod d \Rightarrow \\ (u+v) &\equiv (u'+v') \bmod d \end{aligned}$$

På samme måde fås at :

$$\begin{aligned} d|(u-u') \quad &\text{og} \quad d|(v-v') \Rightarrow \\ d|v'(u-u') \quad &\text{og} \quad d|u(v-v') \Rightarrow \\ d &\mid (v'(u-u')+u(v-v')) \Rightarrow \\ d &\mid (v'u-v'u'+uv-uv')) \Rightarrow \\ d &\mid (uv-u'v') \Rightarrow \\ (uv-u'v') &\equiv 0 \bmod d \Rightarrow \\ u'v' &\equiv uv \bmod d \end{aligned}$$

□

**Sætning 2.17** *Lad $a, m, n \in \mathbb{Z}^3$ og $d \in \mathbb{Z}_+$. Det gælder at :*

$$(a^m \text{ MOD } d)^n \text{ MOD } d = a^{m \cdot n} \text{ MOD } d$$

**Bevis :**

Sætningen følger direkte af regneregler for multiplikation idet $a^{mn}$ MOD $d$ kan skrives som

$$\overbrace{a \cdot a \cdot a \cdot a \cdots a}^{mn} \text{ MOD } d =$$

$$\underbrace{(\overbrace{a \cdot \ldots \cdot a}^{m} \text{ MOD } d) \cdot (\overbrace{a \cdot \ldots \cdot a}^{m} \text{ MOD } d) \ldots (\overbrace{a \cdot \ldots \cdot a}^{m} \text{ MOD } d)}_{n}$$

□

**Sætning 2.18 (Modulær eksponentiering)**
*Beregning af*

$$a^m \text{ MOD } n$$

*kan foregå udelukkende med kvadreringer og multiplikationer, således at ingen mellemresultater bliver større end $n^2$.*

**Bevis :**

Den følgende rekursive algoritme giver ifølge sætning 2.17, det rigtige resultat :

$$\begin{array}{lll} m \text{ lige} & : & a^m \text{ MOD } n = \left(a^{m//2} \text{ MOD } n\right)^2 \text{ MOD } n \\ m \text{ ulige} & : & a^m \text{ MOD } n = (a^{m-1} \text{ MOD } n) \cdot a \text{ MOD } n \\ n = 0 & : & a^0 \text{ MOD } n = 1 \end{array}$$

□

***Eksempel 2.3. Modulær eksponentiering***

*Hvis vi skal beregne udtrykket*

$$3^5 \text{ MOD } 7$$

*kan det gøres på to forskellige måder :*

*Ved at beregne udtrykket og reducere resultatet modulo 7 som vist herunder*

| | | | | | |
|---|---|---|---|---|---|
| 1. | *Kvadrer 3* | : | $3 \cdot 3$ | $= 9$ | $(3^2)$ |
| 2. | *Kvadrer resultatet* | : | $9 \cdot 9$ | $= 81$ | $((3^2)^2 = 3^4)$ |
| 3. | *Multiplicer med 3* | : | $81 \cdot 3$ | $= 243$ | $(3 \cdot 3^4 = 3^5)$ |
| 4. | *Reducer modulo 7* | : | $243 \text{ MOD } 7$ | $= 5$ | |

*eller alternativt at reducere mellemresultaterne modulo 7 :*

| | | | | | |
|---|---|---|---|---|---|
| 1. | *Kvadrer 3* | : | $3 \cdot 3 \text{ MOD } 7$ | $= 2$ | $(9 \text{ MOD } 7)$ |
| 2. | *Kvadrer resultatet* | : | $2 \cdot 2 \text{ MOD } 7$ | $= 4$ | $(4 \text{ MOD } 7)$ |
| 3. | *Multiplicer med 3* | : | $4 \cdot 3 \text{ MOD } 7$ | $= 5$ | $(12 \text{ MOD } 7)$ |

*I det første eksempel kræves det at tallet 243 kan repræsenteres, medens ingen mellemresultater kan blive større end $6 \cdot 6 = 36$ i det andet eksempel.*

**Sætning 2.19** *Lad $p, q \in \mathbb{Z}$ og $\text{SFD}(p, q) = 1$. Da gælder*

$$a \equiv b \bmod p \text{ og } a \equiv b \bmod q \Rightarrow a \equiv b \bmod p \cdot q$$

**Bevis :**

Vi har at

$$a - b \equiv 0 \bmod p \text{ og } a - b \equiv 0 \bmod q$$

som kan skrives

$$p|(a-b) \text{ og } q|(a-b)$$

Da både $p$ og $q$ går op i differensen og da $\mathrm{SFD}(p,q) = 1$, må $p \cdot q$ også gå op, og så får man

$$(p \cdot q)|(a-b) \Rightarrow a \equiv b \bmod p \cdot q$$

□

**Sætning 2.20** *For tal $u, v, m \in \mathbb{Z}$, med $\mathrm{SFD}(u,v,m) = d$ gælder der at*

$$u \equiv v \bmod m \Rightarrow \frac{u}{d} \equiv \frac{v}{d} \bmod \frac{m}{d}$$

**Bevis :**

Vi har at

$$m|(u-v) \Rightarrow \frac{m}{d}|\left(\frac{u}{d} - \frac{v}{d}\right)$$

□

På samme måde kan man bevise følgende

Er $\mathrm{SFD}(m,n) = d$ gælder der følgende :

$$nu \equiv nv \bmod m \Rightarrow u \equiv v \bmod \frac{m}{d}$$

Er $\mathrm{SFD}(u,v,m) = 1$ og $\mathrm{SFD}(u,v) = d$, gælder der at :

$$u \equiv v \bmod m \Rightarrow \frac{u}{d} \equiv \frac{v}{d} \bmod m$$

### 2.4.5 Fermats og Eulers sætninger

To af de vigtigste sætninger indenfor talteori, er Fermats og Eulers sætninger.

**Sætning 2.21 (Eulers sætning)**
*Hvis* $\mathrm{SFD}(a,n) = 1$ *hvor* $a, n \in \mathbb{Z}$ *så gælder :*

$$a^{\phi(n)} \equiv 1 \bmod n$$

**Bevis :**

Ifølge sætning 2.4 gælder at hvis $r = \mathrm{ORD}(a, G_n)$, så er

$$a^r \equiv 1 \bmod n$$

Vi ved endvidere at $r|\mathrm{ORD}(G_n)$, og da $\mathrm{ORD}(G_n) = \phi(n)$ at

$$r|\phi(n) \Leftrightarrow \phi(n) = k \cdot r$$

for et $k \in \mathbb{Z}$.

Man får derfor at :

$$\begin{aligned} a^{\phi(n)} &= a^{kr} \\ &= (a^r)^k \Rightarrow \\ a^{\phi(n)} &\equiv (1)^k \bmod n \Leftrightarrow \\ a^{\phi(n)} &\equiv 1 \bmod n \end{aligned}$$

□

Hvis $n$ er et primtal, er $\phi(n) = n - 1$, og vi får så Fermats sætning :

**Sætning 2.22 (Fermats sætning)**
*Hvis* $p$ *er et primtal og* $a \in \mathbb{Z}$*, er følgende kongruens tilfredsstillet :*

$$a^{p-1} \equiv 1 \bmod p \tag{2.7}$$

**Sætning 2.23** *For $u, r, s \in \mathbb{Z}$ og $m \in \mathbb{N}$, med* $\mathrm{SFD}(u, m) = 1$ *har vi at :*

$$u^r \equiv u^s \bmod m \Leftrightarrow r \equiv s \bmod \phi(m)$$

**Bevis :**

$$u^r \equiv u^s \bmod m \Leftrightarrow u^{r-s} \equiv 1 \bmod m$$

Potensen kan skrives som følger :

$$r - s = k\phi(m) + c \text{ hvor } 0 \leq c < \phi(m)$$

Derfor får man at :

$$\begin{aligned} u^{r-s} &= u^{k\phi(m)+c} \Leftrightarrow \\ u^{r-s} &= (u^{\phi(m)})^k u^c \Leftrightarrow \\ u^{r-s} &\equiv 1^k u^c \bmod m \Leftrightarrow \\ u^{r-s} &\equiv u^c \bmod m \end{aligned}$$

og da $u^{r-s} \equiv 1 \bmod m$ er også $u^c \equiv 1$. Dette er kun opfyldt for $c = 0$. $r - s$ er derfor et multiplum af $\phi(m)$ :

$$r - s = k \cdot \phi(m) \Leftrightarrow r \equiv s \bmod \phi(m)$$

□

## 2.5 Ligninger modulo $n$

### 2.5.1 Den kinesiske restsætning

**Sætning 2.24 (Den kinesiske restsætning)**
*Lad tallene $\{d_1, d_2, ..., d_t\}$ være indbyrdes primiske og sæt*

$$n = d_1 \cdot d_2 \cdots d_t$$

*Lad endvidere $a_1, a_2, \ldots a_t$ være faste tal. Følgende system af kongruenser vil da have en fælles løsning $x$ modulo $n$.*

$$\begin{aligned} x &\equiv a_1 \bmod d_1 \\ x &\equiv a_2 \bmod d_2 \\ x &\equiv a_3 \bmod d_3 \\ &\vdots \\ x &\equiv a_t \bmod d_t \end{aligned} \tag{2.8}$$

**Bevis :**

Da vi ved at $\{d_1, d_2, ..., d_t\}$ er indbyrdes primiske og $n = d_1 \cdot d_2 \cdots d_t$ vil

$$\mathrm{SFD}\left(d_i, \frac{n}{d_i}\right) = 1$$

$\frac{n}{d_i}$ vil derfor have en invers $y_i$ modulo $d_i$ :

$$\frac{n}{d_i} \cdot y_i \equiv 1 \bmod d_i \tag{2.9}$$

For alle $j \neq i$ vil $d_j$ være en faktor i $\frac{n}{d_i}$ og derfor fås

$$\frac{n}{d_i} \cdot y_i \equiv 0 \bmod d_j \tag{2.10}$$

Egenskaberne ved ligningerne 2.9 og 2.10 gør at et udtryk på formen

$$x = a_i \cdot \frac{n}{d_i} \cdot y_i$$

er løsning til den $i$'te ligning men er 0 i de øvrige ligninger. En samlet løsning til ligningerne 2.8 er derfor summen af disse led :

$$x \equiv \frac{n}{d_1} \cdot y_1 \cdot a_1 + \frac{n}{d_2} \cdot y_2 \cdot a_2 + \ldots + \frac{n}{d_t} \cdot y_t \cdot a_t \bmod n \tag{2.11}$$

Indsætter man udtrykket for $x$ i den $i$'te ligning, vil alle led hvor $i \neq j$ være nul og ligning 2.11 reduceres til

$$x \equiv \frac{n}{d_i} \cdot y_i \cdot a_i \bmod d_i \tag{2.12}$$

som ifølge ligning 2.9 bliver til

$$x \equiv a_i \bmod d_i$$

Derfor er $x$ en fælles løsning til alle ligningerne. □

***Eksempel 2.4. Den kinesiske restsætning***

*Vi skal løse de to kongruenser :*

$$\begin{aligned} x &\equiv 1 \bmod 10 \\ x &\equiv 3 \bmod 11 \end{aligned}$$

*Med betegnelserne brugt i det forrige, får man at :*

$$\begin{array}{rcll} d_1 & = & 10 & \qquad d_2 = 11 \\ n & = & d_1 d_2 & \qquad = 110 \\ a_1 & = & 1 & \qquad a_2 = 3 \\ n/d_1 & = & 11 & \qquad n/d_2 = 10 \\ y_1 : 11 \cdot y_1 & \equiv & 1 \bmod 10 \Rightarrow & \qquad y_1 = 1 \\ y_2 : 10 \cdot y_2 & \equiv & 1 \bmod 11 \Rightarrow & \qquad y_2 = 10 \end{array}$$

*Den fælles løsning bliver derfor :*

$$\begin{aligned} x &\equiv 11 \cdot 1 \cdot 1 + 10 \cdot 10 \cdot 3 \bmod 110 \Rightarrow \\ x &\equiv 11 + 300 \bmod 110 \Rightarrow \\ x &\equiv 91 \bmod 110 \end{aligned}$$

**Sætning 2.25** *Der er fire løsninger til ligningssystemet :*

$$\begin{aligned} &(x \equiv a_{11} \bmod d_1 \ \vee \ x \equiv a_{12} \bmod d_1) \wedge \\ &(x \equiv a_{21} \bmod d_2 \ \vee \ x \equiv a_{22} \bmod d_2) \end{aligned}$$

**Bevis :**

Ligningsystemet ovenfor svarer til de fire systemer herunder,

$$\begin{aligned} x \equiv a_{11} \bmod d_1 \ \wedge \ x \equiv a_{21} \bmod d_2 \\ x \equiv a_{11} \bmod d_1 \ \wedge \ x \equiv a_{22} \bmod d_2 \\ x \equiv a_{12} \bmod d_1 \ \wedge \ x \equiv a_{21} \bmod d_2 \\ x \equiv a_{12} \bmod d_1 \ \wedge \ x \equiv a_{22} \bmod d_2 \end{aligned}$$

der hver har en løsning. □

Mere generelt kan det ses at antallet af løsninger til et ligningssystem på nedstående form :

$$\begin{array}{cccccc} (x \equiv a_{11} \bmod d_1 & \vee & \ldots & \vee & x \equiv a_{1l} \bmod d_1) & \wedge \\ \vdots & & & & \vdots & \\ (x \equiv a_{k1} \bmod d_k & \vee & \ldots & \vee & x \equiv a_{kl} \bmod d_k) & \end{array}$$

er $k \cdot l$.

## 2.5.2 Rødder i $x^a \equiv 1 \bmod n$

I det følgende afsnit vil vi se på hvilke og hvor mange tal, der er rødder i kongruensen $x^a \equiv 1 \bmod n$.

**Sætning 2.26** *Hvis $g$ er en frembringer for gruppen $(Z_n, \cdot)$, så vil det gælde, at $g^j$* kun *er rod i*

$$x^a \equiv 1 \bmod n$$

*såfremt*

$$a \cdot j \equiv 0 \bmod \phi(n)$$

*Antallet af rødder i kongruensen er lig med* $\mathrm{SFD}(a, \phi(n))$.

**Bevis :**

Hvis $g$ er en frembringer for $Z_n$ og $x = g^j$ må det gælde, at

$$(g^j)^a \equiv 1 \bmod n$$

hvis og kun hvis $ja$ er et multiplum af

$$\mathrm{ORD}(g, Z_n) = \phi(n)$$

Vi får derfor

$$\begin{aligned} g^{ja} &= g^{k \cdot \phi(n)} \Leftrightarrow \\ g^{ja} &\equiv 1 \bmod p \Rightarrow \\ ja &= k \cdot \phi(n) \Leftrightarrow \\ ja &\equiv 0 \bmod \phi(n) \end{aligned} \tag{2.13}$$

Sætningens første del er hermed bevist.

For at bevise anden del, sætter vi $d = \text{SFD}(a, \phi(n))$. Ligning (2.13) kan så omformes til

$$j \cdot \frac{a}{d} \equiv 0 \bmod \frac{\phi(n)}{d}$$

Dette betyder at hvis der eksisterer en løsning til denne kongruens, skal $j\frac{a}{d}$ være et multiplum af $\frac{\phi(n)}{d}$. Da $\frac{a}{d}$ er primisk med $\frac{\phi(n)}{d}$ må $j$ nødvendigvis være dette multiplum :

$$j = k \cdot \frac{\phi(n)}{d}, k \in Z$$

Alle løsninger til ligning (2.13) skal derfor være af denne form. Da vi regner modulo $\phi(n)$, vil der være $d$ forskellige $j$'er. □

Man ser af sætningen, at når $a$ og $\phi(n)$ er primiske, vil der kun være *en* løsning idet $d = \text{SFD}(a, \phi(n)) = 1$.

***Eksempel 2.5. Rødder i*** $x^a \equiv 1 \bmod n$

*Vi betragter først situationen hvor a og $\phi(n)$ er primiske :*

$$a = 7, n = 13$$

$$x^7 \equiv 1 \bmod 13$$

*har da kun løsningen* 1. *I tilfældet hvor $a|\phi(n)$ er der a løsninger til kongruensen :*

$$a = 4, n = 13$$

$$x^4 \equiv 1 \bmod 13$$

*har løsningerne {1,5,8,12}.*

### 2.5.3 Løsninger til $a^x \equiv 1 \bmod n$

**Sætning 2.27** *Der gælder at*

$$a^x \equiv 1 \bmod n \tag{2.14}$$

*kun har en løsning, hvis a og n er parvist primiske.*

**Bevis :**

For $x = \phi(n)$ er sætningen triviel, idet vi får kongruensen,

$$a^{\phi(n)} \equiv 1 \bmod n$$

som er Eulers sætning.

For $x \neq \phi(n)$, vil vi vise sætningen ved at antage, at ligningen *har* en løsning, selvom $a$ og $n$ ikke er indbyrdes primiske.

Hvis vi sætter $d = \mathrm{SFD}(a, n)$ vil

$$\begin{array}{lll} d|a & \text{og derfor vil} & d|a^x \\ d|n & \text{og derfor vil} & d|kn \text{ for alle } k \in \mathbb{N} \end{array}$$

Kongruensen i formel (2.14) kan skrives som

$$a^x = 1 + kn$$

men da $d|a^x$ og $d|kn$, giver dette en modstrid. $a$ og $n$ må derfor være indbyrdes primiske. □

**Sætning 2.28** *Hvis $a^r \equiv 1 \bmod n$, så vil elementets orden i gruppen være divisor i $r$ :*

$$ORD(a, Z_n)|r$$

**Bevis :**

Fra sætning 2.18 ved vi at

$$\begin{array}{rcl} a^r & \equiv & 1 \bmod n \Leftrightarrow \\ a^r & \equiv & a^0 \bmod n \Leftrightarrow \\ r & \equiv & 0 \bmod \phi(n) \Leftrightarrow \\ r & \equiv & 0 \bmod \mathrm{ORD}(a, Z_n) \Leftrightarrow \\ ORD(a, Z_n) & | & r \end{array}$$

□

# 2.6 Kvadratiske residualer, Legendre's og Jacobi's symboler

## 2.6.1 Kvadratiske residualer

**Definition 2.11 (Kvadratisk residual)**
*Et tal $a$ er et kvadratisk residual modulo $p$, hvor $p$ er et primtal, hvis det er et* kvadrattal *i gruppen $(Z_p, \cdot)$. Tallet skal altså kunne skrives på formen*

$$a \equiv b^2 \bmod p$$

For $p = 13$ vil f.eks $1 \equiv 1^2 \bmod 13$ og $12 \equiv 5^2 \bmod 13$ være kvadratiske residualer.

**Sætning 2.29** *I $(Z_p, \cdot)$ er der $(p-1)/2$ kvadratiske residualer.*

**Bevis :**

For at vise hvor mange kvadratiske residualer der er, skriver man tallene i gruppen op som følger :

$$\{a_1, a_2, a_3, ..., a_{(p-1)/2}, -a_{(p-1)/2}, ..., -a_3, -a_2, -a_1\}$$

hvor f.eks. $-a_1$ er $a_{p-1}$.

Hvis $a \equiv b^2 \bmod p$ vil også $a \equiv (-b)^2 \bmod p$. Da dette gælder for alle tallene i $Z_p$, vil $b$ og $-b$ være de to eneste løsninger til kongruensen $x^2 \equiv a \bmod p$. Da der således er to tal, der rammer hvert kvadratisk residual, vil der være $(p-1)/2$ kvadratiske residualer. Den anden halvdel af tallene i gruppen kaldes for ikke-residualer. □

For $p = 13$ fås tabellen

| $x$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x^2$ MOD 13 | 1 | 4 | 9 | 3 | 12 | 10 | 10 | 12 | 3 | 9 | 4 | 1 |

Tallene i denne tabel viser at $1, 3, 4, 9, 10$ og $12$ er kvadratiske residualer, hvorimod tallene $2, 5, 6, 7, 8$ og $11$ *ikke* er det.

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $2^j$ | 2 | 4 | 8 | 3 | 6 | 12 | 11 | 9 | 5 | 10 | 7 | 1 |
| *residualer* | | $x$ | | $x$ | | $x$ | | $x$ | | $x$ | | $x$ |

**Tabel 2.3:** Kvadratiske residualer i $G_{13}$.

Vi ved at alle tal i restklassegruppen $(Z_p, \cdot)$ kan skrives på formen $g^j \bmod p$, hvor $g$ er en frembringer. Derfor vil de kvadratiske residualer kunne skrives på formen

$$(g^j)^2 \equiv g^{2j} \bmod p$$

De kvadratiske residualer er altså de tal, hvor frembringeren kan skrives i en lige potens. Tilsvarende vil en ulige potens betyde, at tallet ikke er et kvadratisk residual.

I tabel 2.3 har vi brugt frembringeren 2, til at vise hvilke tal der er kvadratiske residualer i $Z_{13}$. Det ses af tabellen, at det kun er de kolonner, hvor $j$ er lige, der indeholder kvadratiske residualer.

### 2.6.2 Legendres symbol

**Definition 2.12 (Legendres symbol)**
*Lad $a \in \mathbb{Z}$ og $p$ være et primtal. Så defineres Legendre's symbol $\left(\frac{a}{p}\right)$, som følger :*

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \textit{hvis } p|a \\ -1 & \textit{hvis } a \textit{ ikke er et kvadratisk residual modulo } p \\ 1 & \textit{hvis } a \textit{ er et kvadratisk residual modulo } p \end{cases}$$

Legendres symbol fortæller altså om $p|a$ eller om $a$ er et kvadratisk residual modulo $p$.

**Sætning 2.30** *Legendres symbol kan beregnes efter følgende formel :*

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \bmod p \tag{2.15}$$

**Bevis :**

Vi deler bevise i to dele afhængigt af, om $p$ går op i a eller ej.

**p|a** Her vil følgende gælde :

$$\left(\frac{a}{p}\right) = 0 \Leftrightarrow p|a \Leftrightarrow a = kp$$

Indsættes dette i formel 2.15 fås :

$$a^{(p-1)/2} \equiv (kp)^{(p-1)/2} \bmod p$$

og da $kp \equiv 0 \bmod p$ har vi

$$a^{(p-1)/2} \equiv 0 \bmod p$$

hvorved vi har vist dette tilfælde.

**p $\not|$a** Vi starter med at kvadrere formel (2.15) :

$$\left(\frac{a}{p}\right)^2 \equiv (a^{(p-1)/2})^2 \bmod p \Leftrightarrow$$

$$1 \equiv a^{p-1} \bmod p$$

$\left(\frac{a}{p}\right)^2$ er 1, da $p \not| a$. Derfor er $\left(\frac{a}{p}\right)$ enten er 1 eller -1. $a^{p-1}$ MOD $p$ er ifølge Fermats sætning også 1.

Vi skal nu blot finde ud af hvornår resultatet er 1 og hvornår det er $-1$.

**Ligningens venstre side $\left(\frac{a}{p}\right)$**

Hvis $g$ er en frembringer i $Z_p$, kan $a$ skrives som :

$$a \equiv g^j \bmod p$$

for et $j$. Da $a$ kun er et kvadratisk hvis $j$ er lige, vil $\left(\frac{a}{p}\right)$ kun være 1, hvis $j$ er lige.

**Ligningens højre side $a^{(p-1)/2}$**

Da $g$ jo er en frembringer, kan $a^{(p-1)/2}$ skrives

$$a^{(p-1)/2} = g^{j(p-1)/2}$$

Da $p-1$ er gruppens orden, vil $g^{j(p-1)/2}$ ifølge sætning 2.28 kun være 1, hvis $j(p-1)/2$ er et multiplum af $p-1$. Det gælder kun hvis $j-2$ er et helt tal, dvs. hvis $j$ er lige.

Da vi altid vil få et resultat der enten er 1 eller $(-1)$, og da begge sider af ligningen er 1 samtidig, må de nødvendigvis være $(-1)$ i alle andre tilfælde.

□

**Sætning 2.31** *Følgende regneregler gælder for Legendre symboler :*

1. $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right)$
2. *For $b$ primisk med $p$ vil :* $\left(\frac{ab^2}{p}\right) = \left(\frac{a}{p}\right)$
3. $\left(\frac{1}{p}\right) = 1$
4. $\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}$

**Bevis :**

1. Denne regel fås ved at benytte formel (2.15), hvorved fås :

$$\left(\frac{ab}{p}\right) \equiv (ab)^{(p-1)/2} \equiv a^{(p-1)/2} \cdot b^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right) \bmod p$$

2. Denne regel vises ved indsættelse i regel 1 :

$$\left(\frac{ab^2}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b^2}{p}\right) = \left(\frac{a}{p}\right) \cdot 1$$

$\left(\frac{b^2}{p}\right)$ er 1, da $b^2$ altid er et kvadratisk residual.

3. At $\left(\frac{1}{p}\right) = 1$ ses af, at $1 = 1^2$ og dermed altid et kvadratisk residual modulo $p$.
4. Resultatet for $\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}$ fås direkte ved at indsætte i formel (2.15) for legendre's symbol.

□

### 2.6.3 Jacobi's symbol

Jacobi's symbol er en generalisering af Legendre's symbol, idet det er udvidet fra kun at omhandle primtal $p$ til også at behandle sammensatte tal $n$.

**Definition 2.13 (Jacobi's symbol)**
*Lad $a \in \mathbb{Z}$ og $n \in \mathbb{Z}$ være et ulige heltal. n's primfaktoropløsning er givet ved :*

$$N = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \dots p_r^{\alpha_r}$$

*Jacobi's symbol $\left(\frac{a}{n}\right)$ er så givet ved produktet af værdien af Legendre's symbol for de enkelte primfaktorer :*

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \cdot \left(\frac{a}{p_2}\right)^{\alpha_2} \cdots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

En vigtig ting at bemærke ved Jacobi's symbol er, at $\left(\frac{a}{n}\right) = 1$ ikke nødvendigvis betyder, at $a$ er et kvadratisk residual modulo $n$.

Af definitionen ses følgende :

- Hvis $\left(\frac{a}{n}\right) = 0$ vil mindst en af primfaktorerne være divisor i $a$. Dette betyder også at $\mathrm{SFD}(a, n) > 1$.
- Hvis $\left(\frac{a}{n}\right) = 1$ vil et lige antal af primfaktorerne være ikke-residualer modulo $n$.
- Hvis $\left(\frac{a}{n}\right) = -1$ vil et ulige antal af primfaktorerne være ikke-residualer modulo $n$.

**Sætning 2.32** *Der gælder følgende :*

$$\left(\frac{a}{n}\right) = \left(\frac{a \text{ MOD } n}{n}\right)$$

# Kapitel 3

# Kompleksitetsteori

Senere i projektet vil vi se at i mange moderne kryptosystemer afhænger sikkerheden i systemet af, hvor kompliceret et givet matematisk problem er at beregne. Redskabet til at måle 'sværhedsgraden' af matematiske funktioner er kompleksitetsteori. Det er baggrunden for at vi i dette kapitel vil beskrive de grundlæggende begreber indenfor denne teori, samt analysere kompleksiteten af nogle af de funktioner som er beskrevet i det forrige kapitel.

## 3.1 Indledende definitioner

Kompleksitetsteori beskæftiger sig med hvor mange ressourcer i form af tid og plads, der skal bruges til at løse et bestemt problem. Grunden til at også pladsen spiller en rolle er, at man let kan forestille sig en problemstilling, der både kan løses med en hurtig men pladskrævende algoritme og med en langsommere og mindre pladskrævende algoritme. Valget af algoritme er afhængig af den pågældende situation. Har man f.eks. kun meget lidt lagerplads til rådighed, er det nødvendigt at anvende den langsommere, men mindre pladskrævende algoritme.

Man vil som regel angive en algoritmes kompleksitet i form af en funktion $f(n)$, hvor $n$ angiver *'størrelsen af inddata'*. Dette kan f.eks. være antallet af inddata eller størrelsesordenen af det tal der skal bruges. Som mål for algoritmens kompleksitet bruges **'store $O$' notationen** ($\mathcal{O}$) :

**Definition 3.1** *En funktion $f(n) = \mathcal{O}(g(n))$, hvis der findes to konstanter $c$ og $n_0$ således at :*

$$|f(n)| \leq c \cdot |g(n)| \text{ for } n \geq n_0$$

Når $f(n)$ er $\mathcal{O}(g(n))$, betyder det altså, at $f(n)$ numerisk set højst er af samme størrelsesorden som $g(n)$ for alle $n > n_0$. Funktionen $g(n)$ kan altså for nogle $n$ godt være mindre end $f(n)$. Omvendt kan man vælge $g(n)$ vilkårligt større end $f(n)$, men normalt vil man vælge det mindst mulige $g(n)$, for at angive størrelsesordenen så præcist som muligt.

***Eksempel 3.1. Store O-notation***

*Hvis*

$$f(n) = 17n + 10$$

*er $f(n)$ $\mathcal{O}(n)$.*

*Dette skyldes, at man kan vælge $c = 18$, så :*

$$f(n) < 18n \text{ for } n > 10$$

*På samme måde ses det, at hvis $f(n)$ er et polynomium :*

$$f(n) = a_t n^t + a_{t-1} n^{t-1} + \ldots + a_1 n + a_0$$

*så vil $f(n) = \mathcal{O}(n^t)$. Alle konstanter og lavereordens led kan altså smides væk.*

Fordelen ved at måle kompleksiteten af matematiske funktioner ved brug af "store $O$" er for det første, at man får et bud på kompleksiteten, der er uafhængigt af den maskine, algoritmen kører på. Samtidig bliver skønnet overskueligt, således at man kan se, hvordan kompleksiteten ændrer sig, når inddata ændres. Hvis f.eks. $f(n)$ er $\mathcal{O}(n^2)$, er det let at se at køretiden højst kan firedobles når $n$ fordobles. Endelig bliver det nemt at inddele algoritmer i klasser efter kompleksitet. Man skelner blandt andet mellem algoritmer med **polynomiel kompleksitet**, hvor $f(n) = \mathcal{O}(n^t)$, og algoritmer med **eksponentiel kompleksitet**, hvor $f(n) = \mathcal{O}(c^n)$ og samtidig må $f(n) \neq= \mathcal{O}(n^t)$

Tabel 3.1, taget fra [16, side 31] giver en oversigt over forskellige kompleksitetsklasser sammen med køretiden for algoritmer i de forskellige klasser. Køretiderne er angivet for en computer, der kan udføre 1 million instruktioner pr sekund.

| Klasse | Kompleksitet | Regne-operationer for $n = 10^6$ | Omtrentlig regnetid |
|---|---|---|---|
| Polynomiel | $\mathcal{O}(n^t)$ | | |
| Konstant | $\mathcal{O}(1)$ | 1 | 1 $\mu$sekund |
| Lineær | $\mathcal{O}(n)$ | $10^6$ | 1 sekund |
| Kvadratisk | $\mathcal{O}(n^2)$ | $10^{12}$ | 10 dage |
| Kubisk | $\mathcal{O}(n^3)$ | $10^{18}$ | 27397 år |
| Eksponentiel | $\mathcal{O}(2^n)$ | $10^{301030}$ | $10^{301016}$ år |

**Tabel 3.1:** Kompleksitets-klasser

## 3.2 Udvalgte store $\mathcal{O}$'er

Med henblik på senere kompleksitetsanalyser vil vi beregne kompleksiteten af nogle af de talteoretiske formler, som vi har gennemgået i det forrige kapitel. Sætningerne er hentet fra [31]. I sætningerne er det forudsat at tallene er skrevet binært[1].

### 3.2.1 De almindelige regneoperationer

Da de fleste regneoperationerne bygger på de fire grundlæggende regnearter, vil vi starte med at beregne deres kompleksitet.

#### Addition og subtraktion

**Sætning 3.1** *To tal hver af længde $k$ bits kan adderes/subtraheres i $\mathcal{O}(k)$ trin.*

**Bevis :**

[1] Bemærk at et binært tal $a$ har cirka $\log_2 a$ cifre.

| Mente | Toptal | Bundtal | Resultat | Ny mente |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Tabel 3.2:** Binær additionstabel med menter

Addition ved brug af *"papir-og-blyant metoden"* af to tal med $k$ binære cifre, foregår ved at man først skriver de to addender op under hinanden. Derefter adderer man tallene søjle for søjle fra højre mod venstre, idet man for hver søjle medregner en eventuel mente. Resultatet af en addition kan findes ved at slå resultatet for hver enkelt søjle op i en tabel som tabel 3.2, hvor alle de mulige kombinationer er repræsenteret.

Addition af to 1 bits tal inklusive en mente kan altså klares med et tabelopslag, tre aflæsninger og to noteringer. Derfor vil kompleksiteten af en sådan addition være konstant $\mathcal{O}(1)$. Da der skal $k$ eller $k+1$ af disse operationer til at addere to $k$ bits tal, har en addition kompleksitet $\mathcal{O}(k)$.

Subtraktion kan udføres på stort set samme måde, blot med en anden tabel. Subtraktion af to $k$ bit tal har derfor også kompleksitet $\mathcal{O}(k)$. □

### Multiplikation og division

**Sætning 3.2** *To tal, hver af længde k bits, kan multipliceres/divideres i* $\mathcal{O}(k^2)$ *trin.*

**Bevis :**

To tal $a$ og $b$, hvor $a \geq b$, med henholdsvis $k$ og $l$ binære cifre, kan multipliceres efter følgende metode :

```
a          :          11101        k bits
b          :           1101        l bits

beregning  :          11101
                     11101
                    11101
                   ----------
resultat   :       101111001
```

**Figur 3.1:** Multiplikation af binære tal

1. Skriv $a$ et antal gange under hinanden, idet $a$ skiftes en plads til venstre for hver bit i $b$. $a$ skal kun skrives hvis den tilsvarende bit i $b$ er 1.
2. Adder de i punkt 1 opstillede tal.

Et eksempel på en multiplikation efter denne metode, er vist i figur 3.1

Vi kommer til at udføre højst $l-1$ additioner, der for de $l-2$ første kun involverer tal af størrelsesorden $k+l$ bits. Ved den sidste addition, kan tallene blive af størrelsesordenen $k+l+1$ bit, på grund af en eventuel mente. Regnetiden bliver derfor :

$$\begin{aligned}(l-2)(k+l)+(k+l+1) &= l^2+lk-2k-2l+k+l+1\\ &= l^2+lk-k-l+1\\ &\leq l^2+lk\\ &\leq 2lk\end{aligned}$$

Den sidste linje i formlen fremkommer, fordi vi har antaget at $k \geq l$. Hvis tallene $a$ og $b$ er af samme størrelsesorden dvs. at $l=k$, får vi derfor kompleksiteten til $\mathcal{O}(k^2)$. Da vi forudsatte at $k \geq l$, vil multiplikation altid være begrænset af $\mathcal{O}(k^2)$.

En division af to tal, der giver kvotient og rest, kan analyseres efter de samme retningslinier, hvis man benytter den sædvanlige divisionsalgoritme, demonstreret i figur 3.2. Denne algoritme har også kompleksiteten $\mathcal{O}(k^2)$. □

```
 Divisor        Dividend          Resultat
-------------------------------------------
 1011011           110         1111 rest 1
  110
 ----------
  101011
   110
 ---------
   10011
    110
  --------
     111
     110
   -------
       1
```

**Figur 3.2:** Division af binære tal

For multiplikation i særdeleshed gælder at der findes langt hurtigere algoritmer. En enkelt er beskrevet i implementeringskapitlet, og [30] beskriver flere andre.

### 3.2.2 Euclids algoritme

**Sætning 3.3** *Euclids algoritme til beregning af den største fælles divisor for to tal :* $\mathrm{SFD}(a,b)$*, hvor* $a > b$*, har kompleksiteten* $\mathcal{O}(log^3 a)$*, eller hvis* $a$ *har* $k$ *binære cifre* $\mathcal{O}(k^3)$*.*

**Bevis :**

Algoritmen går frem på følgende måde :

$$\begin{aligned}
\mathrm{SFD}(a,b) &= \mathrm{SFD}(b, a \bmod b) = \\
\mathrm{SFD}(b,r_1) &= \mathrm{SFD}(r_1, b \bmod r_1) = \\
\mathrm{SFD}(r_1,r_2) &= \mathrm{SFD}(r_2, r_1 \bmod r_2) = \\
\mathrm{SFD}(r_2,r_3) &= \ldots
\end{aligned}$$

Hver beregning af $a$ mod $b$ involverer en division af kompleksitet $\mathcal{O}(k^2) = \mathcal{O}(log_2^2 a)$. Vi skal så blot finde

$$\textit{antallet af skridt} = \textit{antallet af divisioner}$$

Fra beskrivelsen af algoritmen i 2.3.2 ved vi, at resten hele tiden bliver mindre, dvs. $r_{j+1} < r_j$. Vi viser nu, at der yderligere gælder at $r_{j+2} < \frac{1}{2} r_j$ :

- Hvis $r_{j+1} < \frac{1}{2} r_j$ fås :

$$r_{j+2} < r_{j+1} < \frac{1}{2} r_j$$

- Hvis $r_{j+1} > \frac{1}{2} r_j$ fås :

$$\begin{aligned} r_{j+2} = r_j \bmod r_{j+1} &= r_j - r_{j+1} \\ &< r_j - \tfrac{1}{2} r_j = \tfrac{1}{2} r_j \end{aligned}$$

Af dette kan man se, at resten mindst bliver halveret ved hvert andet gennemløb. Dette svarer til at der skæres en bit af tallet ved hvert andet gennemløb. Da der er $k$ bits i det største tal, skal man således højst igennem $2k$ trin, hvilket svarer til $2k$ divisioner. Da hver division har kompleksiteten $\mathcal{O}(k^2)$, bliver kompleksiteten alt i alt :

$$2k \cdot \mathcal{O}(k^2) = \mathcal{O}(k^3) = \mathcal{O}(\log_2^3 a)$$

□

### Euclids modificerede algoritme

I talteorikapitlet, sætning 2.10 viste vi, at hvis $d = \mathrm{SFD}(a, b)$, så findes der et $x, y$ således at :

$$d = x \cdot a + y \cdot b \tag{3.1}$$

Vi viste også, hvordan $x$ og $y$ kunne findes ud fra $a, b$ og $d$, ved hjælp af Euclids algoritme. Denne algoritme har derfor også kompleksiteten $\mathcal{O}(\log_2^3 a)$.

Af samme grund har beregning af den inverse $b$ til et tal $a$ modulo $m$ (sætning 2.14)

$$ab \equiv 1 \bmod m$$

også kompleksiteten $\mathcal{O}(\log_2^3 m)$.

### 3.2.3 Modulær eksponentierings algoritmen

**Sætning 3.4** *Beregning af*

$$a^m \bmod n$$

*kan gennemføres i* $\mathcal{O}(\log_2 m \cdot \log_2^2 n)$ *skridt.*

**Bevis :**

I forrige kapitel beskrev vi en algoritme der beregnede

$$a^m \bmod n$$

ved at beregne $a^2 \bmod n$ et antal gange, svarende til højst antallet af bits i $m$. Vi får altså $\log_2 m$ kvadreringer af tal $a$, som er mindre end $n$ hver med kompleksitet $\mathcal{O}(\log_2^2 n)$. Antallet af divisioner af tal mindre end $n^2$ vil også være $\log_2 m$. Kompleksiteten af hver division er :

$$\mathcal{O}(\log_2^2 n^2) = \mathcal{O}(2\log_2^2 n) = \mathcal{O}(\log_2^2 n)$$

Den samlede kompleksitet bliver derfor :

$$\log_2 m(\mathcal{O}(\log_2^2 n) + \mathcal{O}(\log_2^2 n)) = \mathcal{O}(\log_2 m \cdot \log_2^2 n)$$

□

#### Legendre's og Jacobi's symboler

Legendre's symbol kan beregnes som en potensopløftning modulo $p$. Kompleksiteten af Legendre's symbol er derfor $O(log^3 p)$. Det samme gør sig gældende for Jacobi's symbol. Dette gælder også, når man ikke kender faktoriseringen af $n$. Dette bevis har vi valgt ikke tage med, men interesserede kan finde det i [31].

## 3.3 Afrunding

Den ovenstående gennemgang dækkede kun del af kompleksitetsteorien der er nødvendig for resten af projektet. Kompleksitetsteori omhandler mange andre sider med interesse for kryptografi, såsom NP-problemer, forskellen mellem det beregnelige og det ikke-beregnelige. Se [14,16].

# Kapitel 4

# Primtalstests og faktoriseringsalgoritmer

Primtalstests og faktoriseringsalgoritmer beskæftiger sig med samme problem, men på to forskellige niveauer. Hvor primtalstests giver svar på *om* et tal er et primtal, angiver faktoriseringen yderligere *hvilke* primfaktorer, der indgår i tallet. En faktoriseringsalgoritme giver således mere information om tallet end en primtalstest. Man kan så spørge sig selv, hvorfor primtalstests overhovedet er interessante, når man kan få de samme ting og mere til at vide gennem en faktoriseringsalgoritme ? Der er to grunde. For det første ønsker man ofte kun at vide, *om* et tal er et primtal eller ej. For det andet kan en primtalstest være væsentlig hurtigere end en faktoriseringsalgoritme. De hurtigste algoritmer i dag er det i hvert fald, omend det ikke er bevist, *at* primtalstests generelt er hurtigere end faktoriseringsalgoritmer.

Den enkleste algoritme er den følgende velkendte :

**Algoritme 4.1 Erasthotenes primtals-si**

> Et tal $n$ kan faktoriseres, ved at dividere det med alle primtal mindre end $\sqrt{n}$. Alle de tal der går op er naturligvis faktorer i $n$. Findes der ingen tal som går op, er $n$ et primtal.

Som man kan se, kan denne algoritme bruges til både primtalstest og faktorisering. Det er 'brutalkraft-metoden' indenfor området, og den

er meget langsom. Algoritmen skal igennem

$$\textit{"antallet af primtal"} < \sqrt{n}$$

divisioner, hver med kompleksitet $\mathcal{O}(\log_2^2 \sqrt{n})$, for at checke om $n$ er et primtal. Ifølge afsnit 2.2.3 findes der cirka $\frac{\sqrt{n}}{\log \sqrt{n}}$ primtal mindre end $\sqrt{n}$, så kompleksiteten bliver :

$$\log_2^2 \sqrt{n} \cdot \frac{\sqrt{n}}{\log \sqrt{n}} = \frac{\sqrt{n} \cdot \log^2 \sqrt{n}}{\log \sqrt{n} \cdot \log^2 2} = \mathcal{O}(\sqrt{n} \cdot \log \sqrt{n})$$

Anvendes en computer, der kan udføre en million instruktioner pr. sekund, vil faktorisering og primtalstest af et tal af størrelses-ordenen $10^{50}$ tage op til 300 milliarder år, altså langt over universets levetid. På et år vil et tal af størrelsesordenen $10^{26}$ kunne faktoriseres.

## 4.1 Primtalstests

Normalt skelnes der mellem to typer af tests : de deterministiske og de stokastiske.

- **En deterministisk primtalstest** giver altid et eksakt svar på om tallet er et primtal eller ej. Algoritme 4.1 beskriver således en deterministisk test.
- **En stokastisk primtalstest** giver kun en vis sandsynlighed for at tallet er et primtal, med mindre det afsløres som sammensat. Vil man have en større sandsynlighed, må man udføre testen flere gange.

Et eksempel på en simpel stokastisk test er nedenstående variation af algoritme 4.1.

**Algoritme 4.2 Stokastisk Erasthotenes si**

1. Vælg et tilfældigt tal $b$, hvor $1 < b < n$.
2. Test om $b$ går op i $n$. Gør $b$ det, er $n$ sammensat. Ellers er $n$ måske et primtal.

***Eksempel 4.1. Stokastisk Erasthotenes si***

*Er $n$ i den ovenstående algoritme 100, vælges $b$ i intervallet $2,\ldots,10$. Rammer man et af tallene $2,4,5,10$ afslører testen at 100 er sammensat. I tilfældene $3,6,7,8,9$ siger testen at 100 måske er et primtal. Udføres testen tilstrækkeligt mange gange, med forskellige $b$, vil man på et eller andet tidspunkt finde et af de tal, der går op i 100, og få afsløret at $n$ er sammensat.*

Denne test giver ikke et specielt sikkert resultat, da $n$ kan have ned til 2 divisorer, hvoraf kun den ene ligger i intervallet $[2,\sqrt{n}]$. Sandsynligheden for at ramme en divisor kan derfor være ned til $\frac{1}{\sqrt{n}-1}$. Testen giver derfor en tilsvarende lav sikkerhed for at $n$ er et primtal.

De tre tests der gennemgås i det følgende er alle stokastiske og ligner hinanden en del. De bygger alle på begrebet **et pseudoprimtal**, som er et tal, der i en bestemt test optræder som et primtal uden at være det. De er alle hentet fra [31].

### 4.1.1 Fermats test

Denne første gruppe af pseudoprimtal, kaldes Fermat-pseudoprimtal, og defineres som følger :

**Definition 4.1 (Fermat-pseudoprimtal)**
*Et sammensat tal $n$ kaldes et Fermat-pseudoprimtal med hensyn til en base $b$, hvis* $\mathrm{SFD}(b,n)$ *er 1 og*

$$b^{n-1} \equiv 1 \bmod n \tag{4.1}$$

Hvis $n$ er et primtal svarer definitionen til Fermats sætning, sætning 2.22.

Det er ret få sammensatte tal $n$ der er Fermat-pseudoprimtal. Man har regnet ud at der i intervallet op til $2 \cdot 10^{10}$ findes 882.206.716 primtal, mens der kun er 19.865 tal der er Fermat-pseudoprimtal til base 2 [41].

De følgende sætninger viser med hvilken sandsynlighed, testen kan afgøre, om et tal er et primtal.

**Sætning 4.1** *Hvis $n$ er et Fermat-pseudoprimtal med hensyn til baserne $b_1$ og $b_2$ og*

$$\mathrm{SFD}(b_1, n) = \mathrm{SFD}(b_2, n) = 1$$

*så er $n$ også et Fermat-pseudoprimtal mht. baserne $a = b_1 b_2$ og $c = b_1 b_2^{-1}$.*

**Bevis :**

Vi viser de to tilfælde hver for sig :

$a = b_1 b_2$

$$\left.\begin{aligned} b_1^{n-1} &\equiv 1 \bmod n \\ b_2^{n-1} &\equiv 1 \bmod n \end{aligned}\right\} \Rightarrow$$

$$\begin{aligned} b_1^{n-1} b_2^{n-1} &\equiv 1 \bmod n \Rightarrow \\ (b_1 b_2)^{n-1} &\equiv 1 \bmod n \Rightarrow \\ a^{n-1} &\equiv 1 \bmod n \end{aligned} \tag{4.2}$$

$c = b_1 b_2^{-1}$

$$\begin{aligned} b_2^{n-1} &\equiv 1 \bmod n \Rightarrow \\ b_2^{-(n-1)} b_2^{n-1} &\equiv b_2^{-(n-1)} \bmod n \Rightarrow \\ b_2^0 &\equiv b_2^{-(n-1)} \bmod n \Rightarrow \\ (b_2^{-1})^{(n-1)} &\equiv 1 \bmod n \end{aligned} \tag{4.3}$$

Sammenholder vi 4.2 og 4.3 får vi at

$$\left.\begin{aligned} b_1^{n-1} &\equiv 1 \bmod n \\ (b_2^{-1})^{(n-1)} &\equiv 1 \bmod n \end{aligned}\right\} \Rightarrow$$

$$\begin{aligned} (b_1 b_2^{-1})^{n-1} &\equiv 1 \bmod n \Rightarrow \\ c^{n-1} &\equiv 1 \bmod n \end{aligned} \tag{4.4}$$

□

Ved brug af sætning 4.1, kan vi nu vise en vigtig egenskab ved Fermat testen :

**Sætning 4.2** *Hvis et sammensat tal $n$ ikke er et Fermat-pseudoprimtal for blot en base $b \in Z_n$, vil det heller ikke være det for mindst halvdelen af de mulige baser $b \in Z_n$.*

**Bevis :**

Lad $\{b_1, b_2, \ldots, b_s\}$ være alle de baser hvori $n$ er et Fermat-pseudoprimtal, og lad $b$ være en base hvortil $n$ ikke er et Fermat-pseudoprimtal.

Antag at $n$ er et Fermat-pseudoprimtal til basen $bb_1$. Ifølge sætning 4.1 vil den da også være det for basen $bb_1b_1^{-1} = b$ hvilket giver en modstrid til antagelsen. Derfor kan $n$ ikke være et Fermat-pseudoprimtal til basen $bb_1$. Dette betyder at den heller ikke kan være et Fermat-pseudoprimtal til nogle af baserne $(bb_1), (bb_2), \ldots, (bb_s)$.

Af dette kan vi konkludere at hvis der blot er en base hvortil $n$ *ikke* er et Fermat-pseudoprimtal, så er den ikke et Fermat-pseudoprimtal til mindst 50 % af alle de mulige baser. □

Disse sætninger sætter os nu i stand til at formulere en stokastisk primtalstest :

**Algoritme 4.3 Fermats pseudoprimtalstest**

1. Check at $n$ er ulige og større end 2.
2. Vælg et tilfældigt tal $b$, hvor $1 < b < n$.
3. Beregn SFD$(b, n)$.
   (a) Er den forskellig fra 1 er $n$ sammensat, og så kan man godt stoppe.
   (b) Ellers beregn $b^{n-1}$ mod $n$. Giver dette et tal større end 1, er $n$ sammensat. Ellers er $n$ et primtal med sandsynligheden mindst 50 %, med mindre formel 4.1 holder for alle baser $b$.

Det vil sige at hvis $n$ går frelst igennem testen for $k$ forskellige $b$, er sandsynligheden højst

$$\underbrace{\frac{1}{2} \cdot \frac{1}{2} \cdots \frac{1}{2}}_{k} = \frac{1}{2^k}$$

for at $n$ er sammensat. Som man kan se vil algoritmen langt hurtigere end Erasthotenes stokastiske test, give en høj sikkerhed for at n er et primtal, hvis ikke det afsløres som sammensat.

Algoritmen har imidlertid en uheldig side : Det *med mindre* der optræder rundt omkring. Der findes nemlig sammensatte tal, som klarer Fermat-testen for *alle* baser $b$. Det er de såkaldte **Carmichael-tal**. Der findes altså sammensatte tal, som testen næsten stensikkert vil rubricere som primtal – en ret uheldig defekt for en primtalstest.

### 4.1.2 Solovay-Strassens test

Solovay-Strassens test bygger på **Euler-pseudoprimtallene**. Disse er bedre til primtalstests end Fermat-pseudoprimtallene, da der ikke findes en pendant til Carmichael-tallene.

**Definition 4.2 (Euler-pseudoprimtal)**
*Et ulige sammensat tal $n$ kaldes et Euler-pseudoprimtal til en base $b$ hvis* $\mathrm{SFD}(b,n) = 1$ *og*

$$b^{(n-1)/2} \equiv \left(\frac{b}{n}\right) \bmod n \tag{4.5}$$

Her er $(\frac{b}{n})$ Jacobi's symbol og $b^{(n-1)/2}$ er definitionen på Legendres symbol, hvis $n$ er et primtal.

Sammenhængen mellem Euler- og Fermat-pseudoprimtallene får man ved at kvadrere begge sider af formel 4.5. Så får man Fermat-pseudoprimtalsbetingelsen. Er et tal $n$ et Euler-pseudoprimtal til en base $b$, er det derfor også et Fermat-pseudoprimtal til den samme base.

Sikkerheden i testen er givet ved følgende sætning.

**Sætning 4.3** *For et heltal $b$ og et ulige, sammensat tal $n$ hvor* $\mathrm{SFD}(b,n)$ *er 1, gælder :*

$$b^{(n-1)/2} \equiv \left(\frac{b}{n}\right) \bmod n$$

*for højst 50 % af alle mulige baser $b$.*

**Bevis :**

Vi starter med at vise, at hvis der findes blot en base $b$ hvor formel 4.5 ikke gælder, så gælder den ikke, for mindst halvdelen af alle baser. Derefter viser vi at man for sammensatte $n$ altid kan finde en base hvor formlen afslører at $n$ er sammensat.

Antag at formel 4.5 gælder for $b_1$, men ikke for $b_2$. Den kan da ikke gælde for $b_1 b_2$, idet man så ville få :

$$\begin{aligned} b_1^{(n-1)/2} &\equiv \left(\frac{b_1}{n}\right) \bmod n \\ (b_1 b_2)^{(n-1)/2} &\equiv \left(\frac{b_1 b_2}{n}\right) \bmod n \Rightarrow \\ (b_1)^{(n-1)/2}(b_2)^{(n-1)/2} &\equiv \left(\frac{b_1}{n}\right)\left(\frac{b_2}{n}\right) \bmod n \end{aligned}$$

Fra den øverste og den nederste formel får man :

$$b_2^{(n-1)/2} \equiv \left(\frac{b_2}{n}\right) \bmod n$$

hvad der er i modstrid med påstanden om at formlen ikke gælder for $b_2$.

Heraf ser man, at har man fundet blot en base, hvor formlen ikke gælder, kan man gange den med alle de baser hvor formlen gælder. Derved fås lige så mange baser som afslører hvis $n$ er sammensat.

Vi mangler nu blot at bevise at man, forudsat at $n$ er sammensat, altid kan finde en base $b$ hvor formlen ikke gælder. Det deler vi op i 2 tilfælde :

1. $n$ kan deles med kvadratet på et primtal : $p^2|n$.
2. $n$ er et produkt af primtal i første potens : $n = pp_1p_2 \ldots p_r$.

1. *$n$ kan deles med kvadratet på et primtal*

Vi vælger $b$ som $1 + \frac{n}{p}$. Højresiden i formel 4.5 bliver da

$$\left(\frac{b}{n}\right) = \left(\frac{1+kp}{kp^2}\right) = \left(\frac{1+kp}{p}\right)^2 \left(\frac{1+kp}{k}\right) = 1 \cdot \left(\frac{1}{k}\right) = 1$$

idet vi benytter følgende regler :

- $(\frac{1+kp}{p})^2 = (\pm 1)^2 = 1$ fordi $p \not| (1+kp)$ så Jacobisymbolet giver enten $-1$ eller $+1$.
- $(\frac{1+kp}{k}) = (\frac{1}{k})$ idet vi benytter den regneregel der siger at

$$\left(\frac{a}{n}\right) = \left(\frac{a \text{ MOD } n}{n}\right)$$

Venstresiden af formel 4.5 bliver derimod ikke 1. Dette skyldes, ifølge [31, side 47-48], at $b^j$ kun bliver 1, hvis $p|j$. Dette sker hvis $j$ f.eks vælges til

$$j = (n-1)/2 = (kp^2 - 1)/2$$

$p$ går jo nemlig op i $kp^2$, og kan derfor ikke gå op i $kp^2 - 1$, og således heller ikke i $(kp^2 - 1)/2$.

I dette tilfælde kan man altså være sikker på at finde en base, der afslører $n$'s sammensathed.

2. *$n$ er et produkt af primtal i første potens*

I dette tilfælde viser det sig at være smart at vælge $b$ så

$$\left(\frac{b}{p}\right) = -1 \text{ og } b \equiv 1 \bmod \frac{n}{p} \tag{4.6}$$

Så bliver højresiden af formel 4.5

$$\begin{aligned}\left(\frac{b}{n}\right) &= \left(\frac{b}{p}\right)\left(\frac{b}{n/p}\right) = -\left(\frac{b}{n/p}\right) \\ &= -\left(\frac{1+k(n/p)}{n/p}\right) = -\left(\frac{1}{n/p}\right) = -1\end{aligned}$$

Venstresiden bliver derimod ikke $-1$. Da

$$b \equiv 1 \bmod \frac{n}{p}$$

og dermed

$$b^{(n-1)/2} \equiv 1 \bmod \frac{n}{p}$$

kan

$$b^{(n-1)/2} \not\equiv -1 \bmod n$$

Vi mangler nu kun at bevise, at der eksisterer et sådant $b$.

Omskriver man definitionen af $b$, ligning 4.6 får man ligningerne

$$b \equiv a \bmod p \text{ og } b \equiv 1 \bmod \frac{n}{p}$$

$a$ skal være en ikke-residual, idet det skal gælde at $(\frac{b}{p}) = -1$. Disse to ligninger vil ifølge den kinesiske restsætning have mindst 1 løsning, idet $\mathrm{SFD}(\frac{n}{p}, p) = 1$.

□

Med denne sætning kan vi formulere algoritmen :

**Algoritme 4.4 Solovay-Strassens primtalstest**

1. Check at $n$ er ulige og større end 2.
2. Vælg et $b$ så $0 < b < n$.
3. Beregn $\mathrm{SFD}(b, n)$.
   (a) Er den forskellig fra 1, er $n$ sammensat.
   (b) Er den lig 1, beregnes $b^{(n-1)/2}$ MOD $n$ og $(\frac{b}{n})$ MOD $n$. Er de forskellige er $n$ sammensat. Er de ens, er $n$ et primtal, med sandsynlighed mindst 50 %.

Også her får man, at hvis $n$ klarer testen for $k$ forskellige $b$'er, vil det være sammensat, med sandsynligheden højst $\frac{1}{2^k}$.

### 4.1.3 Miller-Rabins test

Vi er nu nået til den tredje gruppe af pseudoprimtal, de **stærke pseudoprimtal**, der skal bruges i den sidste af de tre stokastiske tests. De defineres således :

**Definition 4.3 (Stærke pseudoprimtal)**
*Lad $n$ være et ulige sammensat tal, og skriv det som $n - 1 = 2^s t$, hvor $t$ er ulige. Da $n - 1$ er lige vil $s$ være større end 0. Lad endelig basen $b$ være et tal i restklassegruppen $Z_n$.*

*Hvis* $\mathrm{SFD}(b, n) = 1$ *og $n$ og $b$ opfylder at*

$$b^t \equiv 1 \bmod n \text{ eller } b^t \equiv -1 \bmod n \tag{4.7}$$

*eller der findes et* $r$, $0 \leq r < s$ *så*

$$b^{2^r t} \equiv -1 \bmod n \tag{4.8}$$

*kaldes* $n$ *et stærkt pseudoprimtal til base* $b$.

Sammenhængen mellem de stærke pseudoprimtal og de to foregående typer pseudoprimtal, ses ved at kvadrere udtrykkene 4.7 og 4.8 en eller flere gange :

$$\begin{aligned} b^{2^r t} &\equiv -1 \Rightarrow \\ b^{2^{r+1} t} &\equiv 1 \Rightarrow \\ &\vdots \\ b^{(n-1)/2} &\equiv 1 \Rightarrow \\ b^{(n-1)} &\equiv 1 \end{aligned}$$

hvor

$$(n-1) = 2^s t \text{ og } (n-1)/2 = 2^{s-1} t$$

Samme ræsonnement gælder ved kvadrering af formel 4.7. Er tallet $n$ derfor et stærkt pseudoprimtal til en base, er det også et Euler-pseudoprimtal og et Fermat pseudoprimtal til den samme base. Der gælder derfor følgende sammenhæng :

$$\begin{array}{ll} & \text{Stærke Pseudoprimtal} \\ \subseteq & \text{Euler Pseudoprimtal} \\ \subseteq & \text{Fermat pseudoprimtal} \end{array}$$

Ligningen

$$x^2 \equiv 1 \bmod n$$

har løsningerne $\pm 1$ hvis $n$ er et primtal. Dette betyder det at man ved beregning af

$$b^u, \; u = (n-1)/2, \;\; (n-1)/4, \;\; \ldots, \;\; (n-1)/2^s$$

vil få $-1$ første gang man ikke får 1, hvis $n$ er et stærkt pseudoprimtal til basen $b$. Får man alt andet end $-1$ ved man, at $n$ er sammensat.

***Eksempel 4.2. Stærke pseudoprimtal***

*Vi har*

$$n = 113, \quad n-1 = 112 = 2^4 \cdot 7 \quad b = 2$$

*og får så :*

$$\begin{array}{lllll} r=3 & 2^{2^3 \cdot 7} & = & 7.020.570.594.000.379.207.936 & \\ & & & 7.020.570.594.000.379.207.936 & \equiv 1 \bmod 113 \\ r=2 & 2^{2^2 \cdot 7} & = & 268435456 \qquad 268435456 & \equiv 1 \bmod 113 \\ r=1 & 2^{2^1 \cdot 7} & = & 16384 \qquad\qquad 16384 & \equiv -1 \bmod 113 \\ r=0 & 2^{2^0 \cdot 7} & = & 128 \qquad\qquad\quad 128 & \equiv 15 \bmod 113 \end{array}$$

*Da man i dette eksempel får -1, første gang man ikke får 1, må 113 være et stærkt pseudoprimtal til basen 2.*

Normalt vil man i udregningen starte nedefra, med $r = 0$, idet man så kan beregne $b^{2^{r+1}t}$ ud fra $b^{2^r t}$ ved at kvadrere :

$$b^{2^{r+1}t} = b^{2 \cdot 2^r t} = (b^{2^r t})^2$$

Man fortsætter således med at kvadrere indtil man får 1 (eller $r$ bliver lig $s$), og checker så at man fik $-1$ gangen før.

Fordelen ved at bruge de stærke pseudoprimtal til primtalstests er, at et tal højst kan være et stærkt pseudoprimtal til 25 % af alle baser, mens det kan være et Euler pseudoprimtal til 50 % af alle baser.

Beviset for dette kræver en hjælpesætning, som vi starter med at vise.

**Sætning 4.4** *Lad $p$ være et primtal og $p \geq 3$, og skriv $p-1$ som $2^{s'}t'$, hvor $t'$ ulige og $s' \geq 1$. Lad $r, t \in Z_+^2$ og $x \in Z_p$.*

*Antallet af løsninger til kongruensen*

$$x^{2^r t} \equiv -1 \bmod p \tag{4.9}$$

*er da*

$$\begin{array}{rl} 0 & \textit{hvis} \quad r \geq s' \\ 2^r \cdot \mathrm{SFD}(t, t') & \textit{hvis} \quad r < s' \end{array}$$

**Bevis :**

Lad $g$ være en frembringer i restklassegruppen $Z_p$ og skriv $x$ som $g^j$, hvor $0 \leq j < p-1$.

Vi ved fra Fermats lille sætning at $g^{p-1} \equiv 1 \bmod p$. Tages kvadratroden på begge sider får vi :

$$g^{(p-1)/2} \equiv -1 \bmod p$$

fordi forskellige potenser af frembringeren op til gruppens orden altid giver forskellige værdier.

Idet $p-1 = 2^{s'}t'$ kan vi omskrive ligning 4.9 :

$$\begin{aligned} x^{2^r t} &\equiv -1 \bmod p \Rightarrow \\ (g^j)^{2^r t} &\equiv g^{(p-1)/2} \bmod 2^{s'}t' + 1 \Rightarrow \\ g^{2^r tj} &\equiv g^{(2^{s'}t')/2} \bmod 2^{s'}t' + 1 \Rightarrow \\ g^{2^r tj} &\equiv g^{2^{s'-1}t'} \bmod 2^{s'}t' + 1 \Rightarrow \\ 2^r tj &\equiv 2^{s'-1}t' \bmod 2^{s'}t' \end{aligned} \tag{4.10}$$

hvor $j$ er den ubekendte. Det sidste skridt fås ifølge sætning 2.23, ved at betragte eksponenterne modulo

$$\mathrm{ORD}(p) = \mathrm{ORD}(2^{s'}t' + 1) = 2^{s'}t'$$

Vi deler op i de to tilfælde $(r > s'-1)$ og $(r \leq s'-1)$

**$r > s'-1$** Her kan ligning 4.10 divideres igennem med $2^{s'-1}$ hvorefter vi får

$$\begin{aligned} 2^{r-(s'-1)}tj &\equiv t' \bmod 2^{s'-(s'-1)}t' \Rightarrow \\ 2^{r-s'+1}tj &\equiv t' \bmod 2t' \end{aligned}$$

Denne kongruens har ingen løsninger, da

*Lige tal* $\not\equiv$ *Ulige tal* mod *lige tal.*

**$r \leq s'-1$** Her divideres ligning 4.10 igennem med

$$\begin{aligned} \mathrm{SFD}(\textit{modulussen, koefficienten til } j) &= \mathrm{SFD}(2^{s'}t', 2^r t) \\ &= 2^r \mathrm{SFD}(t', t) \end{aligned}$$

Lad nu $d = \text{SFD}(t', t)$. Divisionen giver da

$$\frac{t}{d} \cdot j \equiv 2^{s'-r-1} \cdot \frac{t'}{d} \bmod 2^{s'-r} \cdot \frac{t'}{d}$$

Fordi vi har forkortet så langt som vi kan har denne ligning en og kun en løsning modulo $2^{s'-r}\frac{t'}{d}$. Den oprindelige ligning 4.10 har derfor een løsning i hver af $2^r d$ afsnit af restklassegruppen $2^{s'}t'$ hvilket vil sige $2^r d$ løsninger i alt.

□

Vi kan nu vise den følgende sætning :

**Sætning 4.5** *For $n$ ulige og sammensat, vil $n$ højst være et stærkt pseudoprimtal til 25% af alle baser $b$, $0 < b < n$.*

**Bevis :**

Beviset skiller sig ud i tre tilfælde :

1. $n$ er delelig med kvadratet på et primtal $n = k \cdot p^2$.
2. $n$ er et produkt af to primtal $p$ og $q$ : $n = p \cdot q$
3. $n$ er et produkt af mindst 3 forskellige primtal :

$$n = p_1 p_2 \cdots p_k \quad k \geq 3$$

1. $n = k \cdot p^2$.

   Vi viser at $n$ højst kan være et Fermat-pseudoprimtal for $(n-1)/4$ baser $b$ og dermed også højst et stærkt pseudoprimtal for $(n-1)/4$ baser, da

   $n$ ikke et Fermat pseudoprimtal $\Rightarrow$ $n$ ikke et stærkt pseudoprimtal.

   Antag at $b$ er en base hvor $n$ er et Fermat pseudoprimtal :

   $$b^{n-1} \equiv 1 \bmod n$$

   Det medfører at

   $$b^{n-1} \equiv 1 \bmod p^2$$

da

$$\begin{aligned} b^{n-1} &= 1 + k_1 n \text{ og } n = k_2 p^2 \Rightarrow \\ b^{n-1} &= 1 + k_1 k_2 p^2 \Rightarrow \\ b^{n-1} &\equiv 1 \bmod p^2 \end{aligned} \tag{4.11}$$

Vi ser derfor på restklassegruppen $Z_{p^2}$. Denne gruppe har $p^2 - p = p(p-1)$ elementer, og ifølge sætning 2.26 er der derfor

$$d = \text{SFD}(p(p-1), n-1)$$

løsninger til ligning 4.11

Da $p$ går op i $n$, går $p$ ikke op i $n-1$. Derfor er

$$d = \text{SFD}(p(p-1), n-1) = \text{SFD}(p-1, n-1)$$

$d$ kan altså højst blive $p-1$. Antallet af løsninger til ligningen er derfor højst $p-1$. Så brøkdelen af elementer i restklassegruppen $Z_{p^2}$, der tilfredsstiller ligning 4.11 bliver højst

$$\frac{p-1}{p^2} < \frac{p-1}{p^2-1} = \frac{p-1}{(p-1)(p+1)} = \frac{1}{p+1} \leq \frac{1}{4}$$

da $p \geq 3$.

Hermed har vi vist at det er højst $\frac{1}{4}$ af alle $b$ for hvilke $n$ er et pseudoprimtal.

2. $n = pq$.

Vi skriver :

$$\begin{aligned} n-1 &= 2^s t \\ p-1 &= 2^{s'} t' \\ q-1 &= 2^{s''} t'' \end{aligned}$$

hvor $t, t', t''$ er ulige og $s' \leq s''$, hvad der kun har betydning for hvilken af de to faktorer i $n$ vi kalder $p$ og $q$. $s, s', s''$ er alle mindst 1, da $n-1, p-1, q-1$ alle er lige tal.

Et tal $b$ er kun en base hvortil $n$ er et stærkt pseudoprimtal, hvis en af de følgende betingelser er opfyldt :

$$b^t \equiv \pm 1 \bmod p \quad \text{og} \quad b^t \equiv \pm 1 \bmod q \tag{4.12}$$

eller

$$b^{2^r t} \equiv -1 \bmod p \quad \text{og} \quad b^{2^r t} \equiv -1 \bmod q \tag{4.13}$$

for et $r$, hvor $0 \leq r < s$. Ifølge den kinesiske restsætning, sætning 2.24.

Ifølge sætning 2.26 har ligning 4.12 det følgende antal løsninger :

$$\begin{aligned} \mathrm{SFD}(t, p-1)\mathrm{SFD}(t, q-1) &= \mathrm{SFD}(t, 2^{s'}t')\mathrm{SFD}(t, 2^{s''}t'') \\ &= \mathrm{SFD}(t, t')\mathrm{SFD}(t, t'') \\ &\leq t't'' \end{aligned}$$

Bemærk at $2^{s'}$ og $2^{s''}$ forsvinder, da $t, t', t''$ alle er ulige.

Ligning 4.13 har ifølge sætning 4.4 kun løsninger for $r < \min(s', s'') = s'$. Antallet af løsninger for et $r$ er da højst

$$\begin{aligned} 2^r\mathrm{SFD}(t, t') \cdot 2^r\mathrm{SFD}(t, t'') &= 2^{2r}\mathrm{SFD}(t, t') \cdot \mathrm{SFD}(t, t'') \\ &\leq 4^r t't'' \end{aligned}$$

Da $r$ løber fra nul til $s' - 1$ bliver antallet af løsninger i alt

$$4^0 t't'' + 4^1 t't'' + 4^2 t't'' + \ldots + 4^{s'-1} t't'' = t't'' \left( \frac{4^{s'} - 1}{4 - 1} \right)$$

løsninger, jævnfør summationsformlen for kvotientrækker.

Antallet af baser, hvortil $n$ er et stærkt pseudoprimtal er derfor

$$t't'' + t't'' \left( \frac{4^{s'} - 1}{4 - 1} \right)$$

Antallet af mulige baser $b$ er i alt $n$, dvs større end

$$\phi(n) = (p-1)(q-1) = 2^{s'}t' \cdot 2^{s''}t'' = 2^{s'+s''}t't''$$

Derfor bliver brøkdelen af baser, som er stærke pseudoprimtal højst

$$\begin{aligned} \frac{t't'' + t't'' \left( \frac{4^{s'}-1}{4-1} \right)}{2^{s'+s''}t't''} &= \frac{1 + \left( \frac{4^{s'}-1}{4-1} \right)}{2^{s'+s''}} \\ &= 2^{-s'-s''} \left( 1 + \frac{4^{s'} - 1}{4 - 1} \right) \end{aligned}$$

Da vi har antaget at $s'' \geq s'$ ser vi nu specielt på tilfældet $s'' > s'$. Idet vi erstatter $s''$ med $s'+1$ bliver brøken højst

$$\begin{aligned} 2^{-s'-(s'+1)}\left(\frac{3}{3}+\frac{4^{s'}}{3}-\frac{1}{3}\right) &= 2^{-2s'-1}\left(\frac{2}{3}+\frac{4^{s'}}{3}\right) \\ &= \frac{2^{-2s'}}{3}+\frac{2^{-1}}{3} \\ &\leq \frac{2^{-2}}{3}+\frac{2^{-1}}{3} \\ &= \frac{1}{12}+\frac{1}{6} \\ &= \frac{1}{4} \end{aligned}$$

Uligheden gælder fordi $s' \geq 1$, og eksponenten på $2^{-2s'}$ er negativ.

Er $s'' = s'$ må en af ulighederne

$$\mathrm{SFD}(t,t') \leq t' \text{ og } \mathrm{SFD}(t,t'') \leq t''$$

være skarp, idet hvis der gjaldt lighedstegn begge steder, måtte $t'|t$ og $t''|t$, hvorved man ville få

$$\left.\begin{array}{rcl} t' & | & t \\ t'' & | & t \\ t & | & (n-1) \end{array}\right\} \Rightarrow \left\{\begin{array}{rcl} t' & | & (n-1) \\ t'' & | & (n-1) \end{array}\right.$$

Vi gentager lige definitionerne på $n, p$ og $q$ :

$$\begin{array}{rcrcl} n-1 &=& pq-1 &=& 2^s t \\ p-1 &=& &=& 2^{s'}t' \\ q-1 &=& &=& 2^{s''}t'' \end{array}$$

Dette giver

$$\left.\begin{array}{rcl} q-1 &\equiv& 0 \bmod t'' \\ pq-1 &\equiv& 0 \bmod t'' \end{array}\right\} \Rightarrow$$

$$\begin{array}{rcll} pq-1 &\equiv& q-1 \bmod t'' & \Leftrightarrow \\ pq &\equiv& q \bmod t'' & \Leftrightarrow \\ pq-q &\equiv& 0 \bmod t'' & \Leftrightarrow \\ q(p-1) &\equiv& 0 \bmod t'' & \end{array}$$

Da $t'' \not| q$ medfører det at $t''|(p-1)$. På samme måde fås at $t'|(q-1)$. Vi kan nu se at

$$\left.\begin{array}{rcl} t''|(p-1) & \Rightarrow & t''|2^{s'}t' \Rightarrow t''|t' \\ t'|(q-1) & \Rightarrow & t'|2^{s''}t'' \Rightarrow t'|t'' \end{array}\right\} \Rightarrow t' = t''$$

Da vi er i det tilfælde hvor $s' = s''$ kan vi se at

$$p = s't' + 1 = s''t'' + 1 = q$$

$p = q$ implicerer at $n$ er deleligt med et kvadrattal, hvad der er i modstrid med forudsætningerne. Dette er behandlet under tilfælde 1.

Så enten har vi at

$$\mathrm{SFD}(t,t') < t' \text{ eller } \mathrm{SFD}(t,t'') < t''$$

Da $t, t'$ og $t''$ er ulige tal, må den største fælles divisor, der er skarpt mindre være mindre med mindst en faktor 3. Dette ses af følgende :

> Lad $d = \mathrm{SFD}(t,t') < t'$. Så kan $t'$ skrives som $kd$. Ifølge antagelsen om skarp ulighed må $k > 1$. Da $t'$ er ulige, kan $k$ ikke være lige, og må derfor mindst være 3. Derfor er $k \geq 3$ for mindst en af ulighederne.

Derfor kan vi i brøkdelen for tilfældet $s'' > s'$ udskifte $t't''$ med $\frac{1}{3}t't''$ så den bliver til

$$\frac{1}{3}2^{-2s'}\left(\frac{2}{3} + \frac{4^{s'}}{3}\right) = \frac{2^{-2s'+1}}{9} + \frac{1}{9} \leq \frac{1}{18} + \frac{1}{9} = \frac{1}{6} < \frac{1}{4}$$

Hermed er tilfælde 2 gjort færdig.

3. $n = p_1p_2p_3 \ldots p_k$, hvor $k \geq 3$.

   Igen omskriver vi primfaktorerne

$$p_i - 1 = 2^{s_i}t_i$$

   hvor $t_i$ er ulige så alle $s_i \geq 1$, og hvor vi antager at $s_1$ er den mindste af $s_1, s_2 \ldots, s_k$.

   Man kan gennemføre beviset på samme måde, som i tilfælde 2, blot med $k$ ligninger. I stedet for $t't''$ får man $t_1t_2 \cdots t_k$ og i stedet for $s' + s''$ får man $s_1 + s_2 + \ldots + s_k$.

Antallet af baser, hvortil $n$ er et stærkt pseudoprimtal bliver derfor analogt med tilfælde 2

$$t_1 t_2 \cdots t_k + t_1 t_2 \cdots t_k \left( \frac{2^{ks'} - 1}{2^k - 1} \right)$$

hvor k'et erstatter 2-tallet i tilfælde 2.
Antallet af mulige baser er også her

$$\phi(n) = (p_1 - 1)(p_2 - 1) \cdots (p_k - 1) = 2^{s_1 + s_2 + \ldots + s_k}$$

Brøkdelen bliver så

$$\begin{aligned}
2^{-s_1 - \ldots - s_k} \left( 1 + \frac{2^{ks_1} - 1}{2^k - 1} \right) &\leq 2^{-ks_1} \left( \frac{2^k - 1}{2^k - 1} + \frac{2^{ks_1} - 1}{2^k - 1} \right) \\
&= 2^{-ks_1} \left( \frac{2^k - 2}{2^k - 1} + \frac{2^{ks_1}}{2^k - 1} \right) \\
&= 2^{-ks_1} \frac{2^k - 2}{2^k - 1} + \frac{1}{2^k - 1} \\
&\leq 2^{-k} \frac{2^k - 2}{2^k - 1} + \frac{1}{2^k - 1} \\
&= \frac{1 - 2^{1-k} + 1}{2^k - 1} \\
&= \frac{2 - 2^{1-k}}{2^k - 1} \\
&= \frac{2^k 2^{1-k} - 2^{1-k}}{2^k - 1} \\
&= 2^{1-k} \frac{2^k - 1}{2^k - 1} \\
&= 2^{1-k} \\
&\leq \frac{1}{4}
\end{aligned}$$

da $k \geq 3$. Bemærk at brøken kun gælder for $k \geq 3$. Situationen $k = 2$ er omfattet af tilfælde 2. Den første ulighed gælder fordi $s_1$ er den mindste af $s_i$'erne og den anden gælder fordi $s_1 \geq 1$.

□

Vi kan nu angive den sidste af de stokastiske primtals-tests, Miller-Rabins test, som bygger på de stærke pseudo-primtal.

**Algoritme 4.5 Miller-Rabins primtalstest**

1. Check at $n$ er ulige og $n > 2$
2. Vælg et $b$ så $0 < b < n$
3. Beregn $\mathrm{SFD}(b, n)$. Er den større end 1, er tallet sammensat.
4. Beregn $b^{n-1}$ MOD $n$. Er forskellig fra 1, er tallet sammensat.
5. Skriv $n - 1$ som $2^s t$ med $t$ ulige og beregn $b^t \bmod n$.
   (a) Er den forskellig fra $\pm 1$ er $n$ afsløret som sammensat.
   (b) Ellers beregnes

$$(b^t)^2 = b^{2t}, ((b^t)^2)^2 = b^{2^2 t}$$

   modulo $n$, indtil man får $-1$ eller man får

$$b^{2^{r+1}t} \equiv 1 \bmod n$$

   uden at have fået

$$b^{2^r t} \equiv -1 \bmod n$$

   Hvis resultatet er forskelligt fra $-1$ er $n$ sammensat og man stopper ellers er $n$ et primtal med sandsynlighed mindst 75 %.

Da alle beregningerne involveret i testen :

$$\mathrm{SFD}(b, n),\ b^{(n-1)/2} \text{ og } b^2$$

kan beregnes i $\mathcal{O}(\log^3 n)$ trin, kan en test også udføres i $\mathcal{O}(\log^3 n)$ trin. Godkender testen $n$ med $k$ forskellige baser, er $n$ sammensat med sandsynlighed højst

$$\underbrace{\frac{1}{4} \cdot \frac{1}{4} \cdots \frac{1}{4}}_{k} = \frac{1}{4^k}$$

Da $\frac{1}{4^k} = \frac{1}{2^{2k}}$ ser man, at der kun skal testes med halvt så mange baser som i Solovay-Strassens test, for at opnå den samme sikkerhed.

Sandsynligheden 75 % er ikke præcis, men i stedet et generelt minimum. Faktisk har man vist [31, side 121], at der kun findes *et* sammensat $n < 2.5 \cdot 10^{10}$ der ikke er et stærkt pseudoprimtal til en af baserne 2, 3, 5 og 7, nemlig tallet 3.215.031.751. Dette tal afsløres imidlertid allerede af basen 11. Sagt på en anden måde kan man lave en deterministisk primtalstest for alle tal mindre end $2.5 \cdot 10^{10}$, blot ved køre algoritmen igennem for baserne 2,3,5,7 og 11.

Det er værd at bemærke at denne test generelt kan forvandles til en deterministisk test, såfremt man tror på en ikke bevist påstand, kaldet den udvidede Riemann hypotese. Så kan man nemlig vise at der for alle $n$ ulige og sammensat, vil findes mindst en base mindre end $2\log^2 n$, for hvilket $n$ ikke vil klare testen. Det vil sige, at man blot behøver at teste alle baser op til $2\log^2 n$ for at være 100 % sikker på om $n$ er et primtal eller ej. For en yderligere uddybning af dette, se [33].

### 4.1.4 Sammenfatning

Vi har nu gennemgået tre forskellige stokastiske primtalstests. Fermats test, der ikke kunne afsløre Carmichael-tallene som sammensatte, Eulers test, der gav sandsynligheden mindst 50 % for at et Euler-pseudoprimtal ikke var sammensat, og Miller-Rabins test, der gav sandsynligheden mindst 75 % for at et stærkt pseudoprimtal ikke var sammensat.

Der findes imidlertid en endnu hurtigere test end de ovenfor beskrevne, Rumeley-Adlemans test. Det er en deterministisk test, og bygger på helt andre principper end familien af pseudoprimtaltests. Den ville være meget omfattende at beskrive, hvorfor vi har valgt at udelade den. Kompleksiteten af den er

$$\mathcal{O}((\log n)^{c \log\log\log n})$$

hvilket gør den til den hidtil hurtigste test overhovedet. Den er beskrevet i [33,42], som også beskriver en hel del andre primtalstests.

Med primtalstests som de ovenstående og Rumeley-Adlemans test, er det i dag intet problem at teste tal på op mod tusind decimale cifre. Der findes endda test beregnet på specielle typer tal, der kan klare langt større tal.

## 4.2 Faktorisering

Faktoriseringsalgoritmer beskæftiger sig med at finde ikke-trivielle faktorer i et sammensat tal. En faktoriseringsalgoritme er normalt opbygget sådan at den finder *en* faktor i det tal, som skal faktoriseres. Efter at denne faktor er fundet kan tallet divideres med faktoren, hvorefter der fremkommer to nye og mindre tal, som algoritmen kan fortsætte med. Videre forudsætter mange algoritmer at det tal som skal faktoriseres er sammensat, dvs. at det indeholder ikke-trivielle faktorer.

I dette afsnit vil vi beskrive to beslægtede faktoriseringsmetoder : faktor-base metoden og den kvadratiske si.

### 4.2.1 Faktorbase metoden

Denne metode har en kompleksitet, der er sammenlignelig med de hurtigste metoder. Kompleksiteten af metoden er :

$$\mathcal{O}(e^{C\sqrt{\log n \log\log n}})$$

Metoden bygger på følgende sætning :

**Sætning 4.6** *Lad $n$ være et positivt sammensat heltal. Lad $t$ og $s$ være positive heltal.*
*Hvis det gælder at*

$$t^2 \equiv s^2 \bmod n \tag{4.14}$$

$$t \not\equiv \pm s \bmod n \tag{4.15}$$

*Så vil* $\mathrm{SFD}(t \pm s, n)$ *give ikke-trivielle faktorer i $n$.*

**Bevis :**

Kongruensen i formel 4.14 betyder at $n|(t^2 - s^2)$.

Da

$$t^2 - s^2 = (t-s)(t+s) \text{ har vi at } n|(t-s)(t+s) \tag{4.16}$$

Kongruensen 4.15 betyder at

$$n \not| (t-s) \text{ og } n \not| (t+s) \tag{4.17}$$

Fra formel 4.16 kan vi slutte at både $(t-s)$ og $(t+s)$ indeholder faktorer i $n$. Videre kan vi se fra formel 4.17, at ingen af disse faktorer er lig $n$.

Vi kan derfor konkludere, at

$$\text{SFD}(t-s,n) \text{ og } \text{SFD}(t+s,n)$$

giver ikke-trivielle faktorer i $n$. □

Følgende eksempel illustrerer sætning 4.6.

***Eksempel 4.3. Faktorbase-metoden***

*For $n = 24$ har vi at*

$$7^2 \equiv 1 \bmod 24 \text{ og at } 5^2 \equiv 1 \bmod 24$$

*Vi har så*

$$5^2 \equiv 7^2 \bmod 24 \text{ og at } 7 \not\equiv 5 \bmod 24$$

*Vi kan nu finde faktorer i 24 ved*

$$\begin{aligned} \text{SFD}(5+7,24) &= 12 \\ \text{SFD}(5-7,24) &= 2 \end{aligned}$$

Ovenstående eksempel viser at metoden fungerer, men siger ikke noget om hvordan $s$ og $t$ skal vælges, så ligning 4.14 og 4.15 er opfyldt. Dette gøres ved at konstruere en række kvadrattal $b^2$, med kendt primfaktoropløsning modulo $n$. For eksempel :

$$11^2 \equiv 3 \cdot 7 \bmod 100$$

Hvis der blandt de beregnede $b^2$'er kan findes et udsnit, hvor produktet af primfaktoropløsningerne er et kvadratisk residual modulo $n$, har man fundet en løsning til $s^2 \equiv t^2 \bmod n$. Hvis der også gælder at $s \not\equiv t \bmod n$, kan man finde to faktorer ved brug af sætning 4.6, i modsat fald skal man lede videre.

For at sikre at kvadrattallenes primfaktoropløsning rimelig let kan findes, vælges $b^2$ så $b^2$ MOD $n$ giver et lille tal (eventuelt regnet negativt), som med stor sandsynlighed har små primfaktorer.

***Eksempel 4.4. Faktorbase-metoden fortsat***

*Hvis $n$ er 4633, er $\sqrt{n} \approx 68,07$. Vi kan så vælge vores b'er til 67, 68 og 69 :*

$$\begin{array}{rcrrcrl} 67^2 & \equiv & -144 \bmod 4633 & -144 & \equiv & -1 \cdot 2^4 \cdot 3^2 & \bmod 4633 \\ 68^2 & \equiv & -9 \bmod 4633 & -9 & \equiv & -1 \cdot 3^2 & \bmod 4633 \\ 69^2 & \equiv & 128 \bmod 4633 & 128 & \equiv & 2^7 & \bmod 4633 \end{array}$$

*Hvis vi bruger 67 og 68 får vi*

$$67^2 \cdot 68^2 \equiv (-1)^2 \cdot 2^4 \cdot 3^4 \bmod 4633$$

*Heraf kan vi se at både venstre- og højresiden er et kvadratisk residual. Vi sætter nu*

$$\begin{array}{rcrcr} s & = & 67 \cdot 68 & = & 4556 \\ t & = & -1 \cdot 2^2 \cdot 3^2 & = & -36 \end{array}$$

*Nu har vi et $s$ og et $t$ for hvilke formel 4.14 gælder. Hvis også formel 4.15 er opfyldt, kan vi beregne en faktor i tallet.*

*Vi får at*

$$\begin{aligned} s &\equiv -77 \bmod 4633 \\ t &\equiv -36 \bmod 4633 \end{aligned}$$

*Vi kan så beregne en faktor i $n$ ved*

$$\mathrm{SFD}(-77 - (-36), 4633) = 41$$

Vi vil nu prøve at formalisere dette lidt mere.

Vi starter med at definere en **faktoriseringsbase**. Det er en mængde $B = \{p_1, p_2, \ldots, p_h\}$ af primtal, samt evt. tallet -1. Et tal $b_i$ kaldes et B-tal for et givet $n$, hvis $b_i^2$ MOD $n$ kan skrives som et produkt af tal fra $B$. Hvis $B$ for eksempel vælges til $\{-1, 2, 3\}$ og $n$ til 4633, er 67, 68 og 69 fra det tidligere eksempel $B$-tal, idet vi har

$$\begin{array}{rcrl} 67^2 & \equiv & -1 \cdot 2^4 \cdot 3^2 & \bmod 4633 \\ 68^2 & \equiv & -1 \cdot 3^2 & \bmod 4633 \\ 69^2 & \equiv & 2^7 & \bmod 4633 \end{array}$$

Hvis $b_i$ er et $B$-tal, kan vi generelt skrive det som

$$b_i^2 \equiv p_1^{\alpha_{i1}} p_2^{\alpha_{i2}} \dots p_h^{\alpha_{ih}} \bmod n \Leftrightarrow$$

$$b_i^2 \equiv \prod_{j=1}^{h} p_j^{\alpha_{ij}} \bmod n$$

For at konstruere de kvadratiske residualer skal man finde en hel række af $b_i$'er, der er $B$-tal, og ud af dem vælge en delmængde så hvert $p_j$ optræder med en *lige* eksponent.

$$\prod_i b_i^2 \equiv \prod_i \prod_{j=1}^{h} p_j^{\alpha_{ij}} \bmod n \Leftrightarrow$$

$$\prod_i b_i^2 \equiv \prod_{j=1}^{h} \prod_i p_j^{\alpha_{ij}} \bmod n \Leftrightarrow$$

$$\prod_i b_i^2 \equiv \prod_{j=1}^{h} p_j^{\sum_i \alpha_{ij}} \bmod n$$

Dette giver direkte formler for $s^2 \equiv t^2$

$$\begin{array}{rcl@{\qquad}rcl} s^2 & \equiv & \prod_i b_i^2 \bmod n & t^2 & \equiv & \prod_{j=1}^{h} p_j^{\sum_i \alpha_{ij}} \bmod n \\ \Downarrow & & & \Downarrow & & \\ s & \equiv & \prod_i b_i \bmod n & t & \equiv & \prod_{j=1}^{h} p_j^{\sum_i \alpha_{ij}/2} \bmod n \end{array}$$

Hvis vi for hvert $b_i$ skriver $\alpha_{ij}$'erne op som en vektor $\{\alpha_{i1}, \alpha_{i2}, \dots, \alpha_{ih}\}$, søger vi en delmængde af disse vektorer, hvis sumvektor udelukkende består af lige tal. I restklassegruppen modulo 2 ($Z_2$), hvor alle elementer enten er 0 eller 1, vil denne sumvektor af lige tal være nulvektoren. Dette vektorrum kalder vi $F_2^h$.

I ovenstående eksempel, hvor $n = 4633$ får vi vektorerne :

$$\begin{array}{lcll} 67^2 & : & \{1,4,2\} & \text{i } F_2^3 \quad \{1,0,0\} \\ 68^2 & : & \{1,0,2\} & \qquad\quad \{1,0,0\} \\ 69^2 & : & \{0,7,0\} & \qquad\quad \{0,1,0\} \end{array}$$

Heraf ses at man bør vælge 67 og 68 da

$$67^2 \cdot 68^2 \; : \; \{2,4,4\} \quad \text{i } F_2^3 \quad \{0,0,0\}$$

Ved brug af denne vektor-formulering, kan vi hurtigt svare på hvor mange $b_i$ vi skal bruge, for at sikre at nogle af dem kan summere til nul-vektoren i $F_2^h$. Dette svarer nemlig til at vektorerne er lineært afhængige i $F_2^h$. Da $h+1$ vektorer altid er lineært afhængige, skal vi altså højst bruge $h+1$ $b_i$'er der er $B$-tal. Vektorformuleringen giver også metoden til at udvælge de $b_i$'er, som summerer til nul, da det at finde en lineært afhængig delmængde af vektorerne er en standardteknik indenfor lineær algebra.

Den ovenfor beskrevne metode sikrer, at

$$t^2 \equiv s^2 \bmod n$$

men *ikke* at

$$t \not\equiv \pm s \bmod n$$

Hvis $s$ og $t$ skulle være kongruente, konstrueres to nye tal $s$ og $t$ ud fra de samme baser eller nogle nye. Sådan fortsættes indtil begge krav er opfyldt.

Det ovenstående kan opsummeres i faktorbase-algoritmen :

### Algoritme 4.6 Faktorbase-algoritmen

1. Vælg et $y$ så $1 \ll y \ll n$.
2. Generer mængden $B$ som alle primtal mindre end $y$, og eventuelt -1. Denne mængde indeholder $\pi(y)$ elementer.
3. Vælg $\pi(y)+1$ tal $b_i$ mellem 1 og $n$, hvor $b_i^2 \bmod n$, kan skrives som produktet af primtal mindre end $y$.
4. Find et sæt lineært afhængige vektorer ud fra den i trin 3 konstruerede $((\pi(y)+1) \times \pi(y))$ matrix i $F_2^h$, og beregn sumvektoren og husk de tilsvarende $b_i$'er.
5. Konstruer

$$\begin{aligned} t &\equiv \textstyle\prod_i b_i \bmod n \qquad \text{og} \\ s &\equiv \textstyle\prod_{j=1}^{h} p_j^{(\sum_i \alpha_{ij})/2} \bmod n \end{aligned}$$

6. Er $s \equiv \pm t \bmod n$, gentages trin 3 og 4 med nye $b_i$'er, indtil man får

$$s^2 \equiv t^2 \bmod n \text{ og } s \not\equiv \pm t \bmod n$$

Beregn så en divisor i $n$ som $\mathrm{SFD}(s+t, n)$.

***Eksempel 4.5. Faktorbase-algoritmen***

*Vi ønsker at faktorisere* $n = 1829$. *Vi vælger faktoriseringsbasen* $B$ *til at være de første 6 primtal og -1 :* $\{-1, 2, 3, 5, 7, 11, 13\}$.

$b_i$*'erne vælger vi som de tal* $\lfloor\sqrt{k \cdot 1829}\rfloor$ *og* $\lfloor\sqrt{k \cdot 1829}\rfloor + 1$, *der er produkter af de 6 første primtal og -1.*

*For* $k = 1$ *får vi :*

$$\begin{array}{lllllll} \lfloor\sqrt{1829}\rfloor & = 42 & 42^2 & \equiv & -65 & \equiv & -1 \cdot 5 \cdot 13 \bmod 1829 \\ \lfloor\sqrt{1829}\rfloor + 1 & = 43 & 43^2 & \equiv & 20 & \equiv & 2^2 \cdot 5 \bmod 1829 \end{array}$$

*For* $k = 2$ *får vi :*

$$\begin{array}{lllllll} \lfloor\sqrt{3658}\rfloor & = 60 & 60^2 & \equiv & -58 & \equiv & -1 \cdot 2 \cdot 29 \bmod 1829 \\ \lfloor\sqrt{3658}\rfloor + 1 & = 61 & 61^2 & \equiv & 63 & \equiv & 3^2 \cdot 7 \bmod 1829 \end{array}$$

*For* $k = 3$ *får vi :*

$$\begin{array}{lllllll} \lfloor\sqrt{5487}\rfloor & = 74 & 74^2 & \equiv & -11 & \equiv & -1 \cdot 11 \bmod 1829 \\ \lfloor\sqrt{5487}\rfloor + 1 & = 75 & 75^2 & \equiv & 138 & \equiv & 2 \cdot 3 \cdot 23 \bmod 1829 \end{array}$$

*For* $k = 4$ *får vi :*

$$\begin{array}{lllllll} \lfloor\sqrt{7316}\rfloor & = 85 & 85^2 & \equiv & -91 & \equiv & -1 \cdot 7 \cdot 13 \bmod 1829 \\ \lfloor\sqrt{7316}\rfloor + 1 & = 86 & 86^2 & \equiv & 80 & \equiv & 2^4 \cdot 5 \bmod 1829 \end{array}$$

*Tallene 60 og 75 er ikke B-tal, og falder derfor fra. De resterende tal giver så følgende skema, hvor eksponenterne er repræsenteret i* $F_2^6$

| $i$ | $b_i^2$ | $-1$ | 2 | 3 | 5 | 7 | 11 | 13 |
|---|---|---|---|---|---|---|---|---|
| 1 | $42^2$ | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 2 | $43^2$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 3 | $61^2$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 4 | $74^2$ | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 5 | $85^2$ | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6 | $86^2$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

*Vi skal så finde nogle rækker hvis indgange i tabellen summerer til lige tal.*

*Som et første gæt, bruger vi den 2. og 6. række. Dvs* $b_2^2 = 43^2$, *og* $b_6^2 = 86^2$.

$$(43 \cdot 86)^2 \equiv (2^2 \cdot 5)(2^4 \cdot 5) \equiv (2^3 \cdot 5)^2 \bmod 1829$$

*Men da*

$$\begin{aligned} s: \quad 43 \cdot 86 &\equiv 40 \bmod 1829 \quad \textit{og} \\ t: \quad 2^3 \cdot 5 &\equiv 40 \bmod 1829 \end{aligned}$$

*er* $t \equiv s \bmod n$, *så den kombination forkastes.*

*Rækkerne 1, 2, 3 og 5 adderer også til et lige tal, og giver*

$$\begin{aligned} (42 \cdot 43 \cdot 61 \cdot 85)^2 &\equiv (-1 \cdot 5 \cdot 13)(2^2 \cdot 5)(3^2 \cdot 7)(-1 \cdot 7 \cdot 13) \\ &\equiv (-1 \cdot 2 \cdot 3 \cdot 5 \cdot 7 \cdot 13)^2 \bmod 1829 \end{aligned}$$

*Vi har så*

$$\begin{aligned} 42 \cdot 43 \cdot 61 \cdot 85 &\equiv 1459 \bmod 1829 \textit{ og} \\ -1 \cdot 2 \cdot 3 \cdot 5 \cdot 7 \cdot 13 &\equiv -901 \bmod 1829 \end{aligned}$$

*derfor fås*

$$\mathrm{SFD}((1459 + 901), 1829) = 59$$

*59 er altså en faktor i 1829.*

$$1829/59 = 31$$

### 4.2.2 Kompleksitet af Faktor-base metoden

I det følgende vil vi give et groft skøn over faktor-base metodens kompleksitet. Beviserne bygger på nogle sætninger, taget fra [31], som vi her vil angive uden bevis.

**Sætning 4.7** *Stirlings formel*

$$\log(n!) \approx n \log n - n$$

**Sætning 4.8** *Givet to positive tal* $n$ *og* $u$. *Antallet af måder hvorpå* $n$ *ordnede* $\alpha_j$ *kan udvælges, så det gælder at*

$$\sum_{j=1}^{n} \alpha_j \leq u$$

*er givet ved den følgende binomial-koefficient :*

$$\binom{\lfloor u \rfloor + n}{n} = \frac{(\lfloor u \rfloor + n)!}{n!(\lfloor u \rfloor !)}$$

Første trin i beregning af kompleksiteten, er at vurdere hvor svært det er at generere $b_i$'erne. Dvs beregne hvor stor sandsynlighed der er for at et tilfældigt tal mindre end $n$, er et produkt af primtal mindre end $y$, det tal der i algoritmen er valgt som øvre grænse for tallene i basen B. Sandsynligheden er givet ved forholdet mellem to størrelser :

1. Antallet af tal mindre end $n$, som er produktet af primtal mindre end $y$. Dette kaldes for $\Psi(n, y)$.

2. Antallet af mulige tal : $n$.

**Sætning 4.9** *Sandsynligheden for at et tilfældigt valgt tal mindre end $n$ er produktet af primtal mindre end $y$, er givet ved*

$$\frac{\Psi(n,y)}{n} \approx u^{-u} \text{ hvor } u = \frac{\log n}{\log y}$$

**Bevis :**

Alle tal mindre end $n$, som er produktet af primtal mindre end $y$, kan skrives som

$$\prod_{j=1}^{\pi(y)} p_j^{\alpha_j} \leq n \tag{4.18}$$

hvor $p_j < y$. $\Psi(n, y)$ er derfor antallet af løsninger til uligheden i formel 4.18, med $\alpha_j$'erne som de ubekendte.

For at løse uligheden, tager vi først logaritmen på begge sider og får :

$$\log(\prod_{j=1}^{\pi(y)} p_j^{\alpha_j}) \leq \log n \Rightarrow \sum_{j=1}^{\pi(y)} \alpha_j \log p_j \leq \log n$$

Da de fleste primtal $p_j$ mindre end $y$ er af stort set samme størrelsesorden som $y$, simplificerer vi nu udtrykket ved at erstatte $p_j$ med $y$, og får

$$\sum_{j=1}^{\pi(y)} \alpha_j \log y \leq \log n \Rightarrow \sum_{j=1}^{\pi(y)} \alpha_j \leq \frac{\log n}{\log y} \tag{4.19}$$

Kalder vi $\frac{\log n}{\log y}$ for $u$, bliver ligning 4.19 til :

$$\sum_{j=1}^{\pi(y)} \alpha_j \leq u$$

Vi foretager nu vores næste simplifikation, idet vi erstatter $\pi(y)$ med $y$. Denne simplifikation er ikke triviel, men ifølge [31, side 137-138], gør det ikke nogen forskel i den sidste ende.

Vi får så

$$\sum_{j=1}^{y} \alpha_j \leq u \tag{4.20}$$

Ifølge sætning 4.8 er antallet af løsninger til ulighed 4.20

$$\binom{\lfloor u \rfloor + y}{y}$$

Vi har altså at

$$\Psi(n,y) \approx \binom{\lfloor u \rfloor + y}{y}$$

Vi kan nu beregne brøken :

$$\frac{\Psi(n,y)}{n}$$

Vi tager først logaritmen og får :

$$\log\left(\frac{\Psi(n,y)}{n}\right) = \log \Psi(n,y) - \log n$$

Da

$$u = \frac{\log n}{\log y} \Leftrightarrow \log n = u \log y$$

efter definitionen på $u$, og da

$$\Psi(n,y) \approx \binom{\lfloor u \rfloor + y}{y} = \frac{(\lfloor u \rfloor + y)!}{y! \lfloor u \rfloor !}$$

får vi

$$\begin{aligned}
\log \Psi(n,y) \; - \; & \log n \\
\approx \; & \log\left(\frac{(\lfloor u \rfloor + y)!}{y! \lfloor u \rfloor !}\right) - u \log y \\
= \; & \log((\lfloor u \rfloor + y)!) - \log(y! \lfloor u \rfloor !) - u \log y \\
= \; & \log((\lfloor u \rfloor + y)!) - \log(y!) - \log(\lfloor u \rfloor !) - u \log y
\end{aligned}$$

Vi bruger nu Stirlings formel på de tre første led i udtrykket, og får

$$\overbrace{(\lfloor u\rfloor + y)\log(\lfloor u\rfloor + y) - (\lfloor u\rfloor + y)}^{\log((\lfloor u\rfloor+y)!)} - \underbrace{y\log y + y}_{\log(y!)} - \underbrace{\lfloor u\rfloor \log\lfloor u\rfloor + \lfloor u\rfloor}_{\log(\lfloor u\rfloor!)} - u\log y$$

Vi gør nu yderligere to simplifikationer. Vi erstatter $\lfloor u\rfloor$ med $u$, og $\log(u+y)$ med $\log y$. Det sidste kan vi tillade os, da $u \ll y$. Vi får

$$u\log y + y\log y - u - y - u\log u + u - y\log y + y - u\log y$$

$$= -u\log u$$

alt i alt giver dette at

$$\log\left(\frac{\Psi(n,y)}{n}\right) \approx -u\log u$$

eller

$$\frac{\Psi(n,y)}{n} \approx u^{-u}$$

□

***Eksempel 4.6. Kompleksitet af faktorbase-algoritmen***

*Lad $n$ være $10^{48}$, og $y$ være $10^6$, så er*

$$u = \frac{\log 10^{48}}{\log 10^6} = \frac{48\log 10}{6\log 10} = \frac{48}{6} = 8$$

*Sandsynligheden bliver*

$$u^{-u} = 8^{-8} \approx 6\cdot 10^{-8}$$

Vi kan nu vurdere faktor-base algoritmens kompleksitet. Vi vil antage at faktorbasen består af de første $h = \pi(y)$ primtal, dvs. alle primtal mindre end $y$, og at den ikke indeholder $-1$.

**Sætning 4.10** *Kompleksiteten af faktorbase-algoritmen er*

$$\mathcal{O}(e^{C\sqrt{\log n \log\log n}})$$

**Bevis :**

Der er tre beregningsmæssigt 'tunge' trin i algoritmen. Det er

1. Generering af mængden $B$.
2. Dannelse af de $\pi(y)+1$ tal $b_i$ der er $B$-tal.
3. Beregning af et sæt lineært afhængigt vektorer.

**Trin 1** behøver ikke at blive foretaget hver gang algoritmen skal køre, men kan laves en gang for alle. Det vil vi derfor se bort fra her.

I både trin 2 og 3, spiller antallet af $b_i$ vi skal vælge en rolle. Dette antal afhænger imidlertid af $y$, og vi har endnu ikke beskrevet hvordan vi vælger $y$. Hvis vi vælger $y$ lille, bliver det svært at finde $b_i$'erne, da intervallet $]1, y[$ som faktorerne i $b_i$ skal ligge i, bliver mindre. Til gengæld bliver matrix-reduktionen hurtig, da matricen bliver lille. Omvendt gør et stort $y$ det nemmere at finde $b_i$'erne, mens matrix-reduktionen bliver sværere. Der er derfor ikke noget umiddelbart bud på størrelsen af $y$. Vi opstiller derfor først kompleksiteten som en funktion af både $y$ og $n$, og minimaliserer derefter det fundne udtryk med hensyn til $y$.

**Trin 2** indbefatter at der skal findes $(\pi(y)+1)b_i$'er. Sætning 4.9 viser at der skal $u^u$ forsøg til at finde et $b_i$. Der skal derfor $u^u(\pi(y)+1)$ forsøg til at finde $(\pi(y)+1)$ $b_i$'er. For at vurdere kompleksiteten af trin 2, mangler vi at finde ud af, dels hvor lang tid det tager at generere et $b_i$, dels hvor lang tid det tager at beregne $b_i^2$ MOD $n$, og dels hvor lang tid det tager at teste om *et* $b_i^2$, kan opløses i faktorer fra faktorbasen.

Da det tager en konstant tid at generere 1 bit af et tal, bliver kompleksiteten af at generere et $b_i < n$ $\mathcal{O}(r)$, idet $n$ har $r$ cifre. Beregningen af $b_i^2$ mod $n$, har som tidligere vist kompleksiteten $\mathcal{O}(r^2)$.

Vi skal så teste om $b_i^2$ mod $n$ kun består af faktorer mindre end $y$, og for de $b_i$ som gør det, notere faktorerne. Vi skal

med andre ord faktorisere $b_i^2 \bmod n$. En simpel måde at gøre det på, er ved at benytte Erasthotenes si. Vi erindrer at dette gøres ved at dele $b_i^2 \bmod n$ med 2 og alle ulige tal mellem 2 og $y$ i rækkefølge, og undervejs notere faktorerne og hvor mange gange de optræder. Da en division af et tal med højst $r$ bits, med et tal med højst $s$ bits, hvor $s$ er antallet af bits i $y$, tager $\mathcal{O}(rs)$ trin, tager selve testen $\mathcal{O}(rsy)$ trin. Kompleksiteten af at udvælge et $b_i^2$ er da

$$\begin{aligned} u^u(\pi(y)+1)(\mathcal{O}(r) &+ \mathcal{O}(r^2)+\mathcal{O}(rsy)) \\ &= u^u(\pi(y)+1)\mathcal{O}(rsy) \end{aligned} \quad (4.21)$$

idet $\mathcal{O}(r) < \mathcal{O}(r^2)$ og $r \ll y \Rightarrow \mathcal{O}(r^2) < \mathcal{O}(rsy)$.
Da $\pi(y) \approx y/\log y = \mathcal{O}(y/s)$, bliver formel 4.21 til

$$\begin{aligned} u^u(\pi(y)+1)\mathcal{O}(rsy) &\approx u^u\mathcal{O}(y/s)\mathcal{O}(rsy) \\ &= u^u\mathcal{O}(ry^2) \end{aligned}$$

Trin 2 tager altså $u^u\mathcal{O}(ry^2)$ bit-operationer.

**Trin 3** i algoritmen involverer matrix-reduktion og beregning af $s$ og $t$ mod $n$. Begge dele kan ifølge [31] gennemføres i polynomiel tid. Trin 3 kan derfor gennemføres med

$$\mathcal{O}(y^j r^h)$$

bit-operationer for passende valg af $j$ og $h$.

Vi kan nu vurdere den samlede kompleksitet. Hver gang trin 2 og 3 i algoritmen gennemløbes, er der, ifølge [31] mindst 50 % sandsynlighed for, at de konstruerede $s$ og $t$ opfylder betingelserne i sætning 4.6. Det betyder at hvis vi stiller os tilfredse med en sandsynlighed på $(1 - 2^{-50})$, for at finde en faktor, skal vi gennemløbe algoritmen 50 gange.

Vi får så følgende tidsestimat :

$$\mathcal{O}(50(u^u r y^2 + y^j r^h)) = \mathcal{O}(r^h u^u y^j) \quad (4.22)$$

Da $\log y \approx s$ er $y^j \approx e^{js}$ og $u$ er $\frac{\log n}{\log y} \approx \frac{r}{s}$ får vi derfor

$$\mathcal{O}(r^h u^u y^j) = \mathcal{O}(r^h \left(\frac{r}{s}\right)^{\left(\frac{r}{s}\right)} e^{js}) \quad (4.23)$$

for passende valg af $h$ og $j$.

Vi mangler nu blot at minimalisere dette udtryk med hensyn til $y$, dvs. med hensyn til $s$, hvad der betyder at vi skal minimalisere

$$\left(\frac{r}{s}\right)^{\left(\frac{r}{s}\right)} e^{js} \tag{4.24}$$

Vi tager først logaritmen til formel 4.24, og minimaliserer så den.

$$\left(\frac{r}{s}\right) \log \left(\frac{r}{s}\right) + js$$

For at finde minimum differentierer vi og finder rødderne :

$$\begin{aligned}
\frac{d}{ds}\left(\frac{r}{s}\log\frac{r}{s} + js\right) &= 0 \Rightarrow \\
\frac{-r}{s^2}\left(\log\frac{r}{s} + 1\right) + j &= 0 \Rightarrow \\
\frac{-r}{s^2}\left(\log\frac{r}{s} + c\cdot\log\frac{r}{s}\right) + j &= 0 \Rightarrow \\
c'\cdot\frac{-r}{s^2}\log\frac{r}{s} + j &\approx 0 \Rightarrow \\
js &\approx c'\cdot\frac{r}{s}\log\frac{r}{s} \Rightarrow \\
e^{js} &= \frac{r^{\frac{rc'}{s}}}{s}
\end{aligned} \tag{4.25}$$

Kilde [31] ser bort fra 1-tallet mellem linie 2 og 3. Vi mener ikke selv, at det virker helt ukritisk, da det to led 1 og $\log\frac{r}{s}$ er cirka lige store, hvorfor vi vurderer det med udtrykket $c\cdot\log\frac{r}{s}$

Ligning 4.25 betyder at de to størrelser $js$ og $\frac{rc'}{s}\log\frac{r}{s}$ skal være ca lige store for at udtrykket er minimaliseret.

Inden vi regner videre på ligning 4.23, vil vi finde $s$ som en funktion af $r$. Fra 4.25 får vi :

$$\begin{aligned}
js^2 &\approx c'r\log\frac{r}{s} \Rightarrow \\
s^2 &\approx c'\cdot\frac{r}{j}(\log r - \log s)
\end{aligned}$$

Da $\log s \ll \log r$, smider vi det væk, og får :

$$s \approx \sqrt{c'\cdot\frac{r}{j}\log r} \tag{4.26}$$

Dette udtryk for $s$ bruger vi til at bestemme tidsforbruget, som vi tidligere fik til

$$\mathcal{O}(r^h \left(\frac{r}{s}\right)^{\left(\frac{r}{s}\right)} e^{js})$$

Vi viste så at,

$$\begin{aligned} \frac{r^{\frac{rc'}{s}}}{s} &\approx e^{js} \Leftrightarrow \\ \frac{r^{\frac{r}{s}}}{s} &\approx e^{\frac{js}{c'}} \end{aligned}$$

hvilket betyder, at vi kan sætte $\left(\frac{r}{s}\right)^{\left(\frac{r}{s}\right)} e^{js} = e^{c''js}$, hvorefter tidsforbruget bliver

$$\mathcal{O}(r^h e^{c''js})$$

Når vi så indsætter udtrykket for $s$ fra formel 4.26 får vi

$$\begin{aligned} \mathcal{O}\left(e^{c''j\sqrt{(r/j)\log r}} \cdot r^h\right) &= \mathcal{O}\left(e^{\frac{c''j}{\sqrt{j}}\sqrt{r\log r}} \cdot r^h\right) \\ &= \mathcal{O}\left(e^{c''\sqrt{j}\sqrt{r\log r}} \cdot r^h\right) \end{aligned}$$

Sætter vi $c''\sqrt{j} = k$ får vi

$$\mathcal{O}\left(e^{k\sqrt{r\log r}} \cdot r^h\right)$$

$r^h$ er $\mathcal{O}(e^{\sqrt{r\log r}})$, så vi får

$$\mathcal{O}\left(e^{k\sqrt{r\log r}} \cdot r^h\right) = \mathcal{O}\left(e^{2k\sqrt{r\log r}}\right) = \mathcal{O}\left(e^{k'\sqrt{r\log r}}\right)$$

Da $r \approx \log n$ kan dette også skrives som

$$\mathcal{O}\left(e^{k'\sqrt{\log n \log\log n}}\right)$$

□

Hermed har vi givet et groft skøn over Faktorbase algoritmens kompleksitet. Beviset indeholder en del simplifikationer, der gør at skønnet er behæftet med usikkerhed.

### 4.2.3 Den kvadratiske si

Den kvadratiske si er den hurtigste faktoriseringsmetode, der findes i dag, og implementeringer bygget over den, har rekorden for faktorisering af store tal. Metoden er en videreudvikling af faktor-base metoden, som specielt forbedrer måden hvorpå $b_i$'erne genereres. I stedet for at prøve at faktorisere *alle* de genererede $b_i$'er, genererer man kun dem, der kan opløses i primfaktorer fra basen. Således spares en masse forgæves faktoriseringer. Vores korte gennemgang er hentet fra [43].

Man vælger som før en base bestående af -1, 2 og primtallene $p_i$ op til en grænse $y$. Selve $b_i$ fås ved

$$b_i = \lfloor \sqrt{n} \rfloor + x \tag{4.27}$$

for forskellige valg af $x$ i et interval omkring 0.

De tal i intervallet $[-n/2, n/2]$, som er kongruente med $b_i^2$ MOD $n$ fås af funktionen

$$Q(x) = (x + \lfloor \sqrt{n} \rfloor)^2 - n \tag{4.28}$$

som vil give den principale rest $r$, sålænge $x$ er mindre end $(\sqrt{2}+1)\sqrt{n}$, hvilket ses af følgende argument :

$$\begin{array}{lcll} r & \equiv & b_i^2 \bmod n & \Leftrightarrow \\ r & \equiv & (\lfloor \sqrt{n} \rfloor + x)^2 \bmod n & \Leftrightarrow \\ r & = & (\lfloor \sqrt{n} \rfloor + x)^2 + k \cdot n & \end{array}$$

og da den principale rest $r$ er mindre end $n$ og funktionen $Q$ i formel 4.28 har konstanten $k = -1$, får vi uligheden

$$\begin{array}{lcll} (\lfloor \sqrt{n} \rfloor + x)^2 - n & < & n & \Leftrightarrow \\ (\lfloor \sqrt{n} \rfloor + x)^2 & < & 2n & \Leftrightarrow \\ \lfloor \sqrt{n} \rfloor + x & \leq & \lfloor \sqrt{2n} \rfloor & \Leftrightarrow \\ x & \leq & (1+\sqrt{2})\lfloor \sqrt{n} \rfloor & \end{array}$$

Funktionen $Q(x)$ har den egenskab at hvis $p_i$ går op i $Q(x)$, så vil $p_i$ gå op i $Q(x + kp_i)$ for alle $k$, da man har

$$\begin{aligned} Q(x + kp_i) &= ((x + kp_i) + \lfloor \sqrt{n} \rfloor)^2 - n \\ &= (x + kp_i)^2 + \lfloor \sqrt{n} \rfloor^2 + 2(x + kp_i)\lfloor \sqrt{n} \rfloor - n \\ &= x^2 + (kp_i)^2 + 2xkp_i + \lfloor \sqrt{n} \rfloor^2 \\ &\quad +2x\lfloor \sqrt{n} \rfloor + 2kp_i\lfloor \sqrt{n} \rfloor - n \end{aligned}$$

$$\begin{aligned} &= (x^2 + \lfloor\sqrt{n}\rfloor^2 + 2x\lfloor\sqrt{n}\rfloor - n) \\ &\quad +(kp_i)^2 + 2xkp_i + 2kp_i\lfloor\sqrt{n}\rfloor \\ &= Q(x) + (kp_i + 2x + 2\lfloor\sqrt{n}\rfloor) \end{aligned}$$

og $p_i$ går op i det sidste udtryk.

Det betyder, at har man fundet *en* base, hvor $p_i$ er divisor, har man en hel mængde hvor den også er det, fordi hvis man bare har et $x_0$ hvor $p_i|Q(x_0)$, vil også $p_i|Q(x_0 + kp_i)$, hvilket vil sige at $p_i$ også er divisor i alle andre $x$ med afstand $kp_i$ til dette $x_0$.

I den kvadratiske si gør man så følgende :

Man tager alle $x$ i et interval, f.eks.

$$x = -1000, \ldots, -1, 0, 1, 2, \ldots, 1000$$

og beregner $Q(x)$ for alle disse $x$ og gemmer værdierne, som er kongruente med de forskellige $b_i^2$ i en tabel.

Så løbes basen $B$ igennem. For hvert primtal $p_i$ i basen finder man et $x_0$, hvor $p_i|Q(x_0)$. Når et sådant findes, divideres $p_i$ igennem $Q(x)$ for dette $x_0$ og for alle de $x = x_0 + kp_i$, der ligger i $x$-intervallet.

Efter at have løbet hele $B$ igennem vil nogle af værdierne i tabellen være reduceret til 1. Det er disse tal, der er kun består af primfaktorer fra basen $B$ og dermed de tal, der er $B$-tal. Man kan så vælge nye intervaller indtil man har fundet nok $B$-tal.

Grunden til at metoden kaldes *den kvadratiske si*, er at man så at sige 'sier' $B$-tallene fra, lidt på samme måde som Erasthotenes gjorde.

### *Eksempel 4.7. Den kvadratiske si*

*Vi vil faktorisere 5069, og vælger basen* $\{-1, 2, 5, 7, 11, 13\}$.

*Vi prøver med* $x = \{-8, -7, \ldots, -1, 0, 1, \ldots, 7, 8\}$, *og får tabel 4.1. De relevante tal er dem uden rest :*

$$63, 69, 70, 71, 73, 80$$

| $x$ | $b_i$ $x+\sqrt{n}$ | $b_i^2$ MOD $n$ $Q(x)$ | $b_i^2$ opløst som faktorer i $B$ | | | | | | Rest |
|---|---|---|---|---|---|---|---|---|---|
| −8 | 63 | −1100 | −1 | $2^2$ | $5^2$ | | 11 | | 0 |
| −7 | 64 | −973 | −1 | | | 7 | | | 139 |
| −6 | 65 | −844 | −1 | $2^2$ | | | | | 211 |
| −5 | 66 | −713 | −1 | | | | | | 713 |
| −4 | 67 | −580 | −1 | $2^2$ | 5 | | | | 29 |
| −3 | 68 | −445 | −1 | | 5 | | | | 89 |
| −2 | 69 | −308 | −1 | $2^2$ | | 7 | 11 | | 0 |
| −1 | 70 | −169 | −1 | | | | | $13^2$ | 0 |
| 0 | 71 | −28 | −1 | $2^2$ | | 7 | | | 0 |
| 1 | 72 | 115 | | | 5 | | | | 23 |
| 2 | 73 | 260 | | $2^2$ | 5 | | | 13 | 0 |
| 3 | 74 | 407 | | | | | 11 | | 37 |
| 4 | 75 | 556 | | $2^2$ | | | | | 139 |
| 5 | 76 | 707 | | | | 7 | | | 101 |
| 6 | 77 | 860 | | $2^2$ | 5 | | | | 43 |
| 7 | 78 | 1015 | | | 5 | 7 | | | 29 |
| 8 | 79 | 1172 | | $2^2$ | | | | | 293 |
| 9 | 80 | 1331 | | | | | $11^3$ | | 0 |

**Tabel 4.1:** Faktoriseringstabel til den kvadratiske si

*Herfra fortsætter metoden som faktorbase-metoden, og man får f.eks :*

$$\begin{aligned} 80^2 \cdot 70^2 \cdot 63^2 &\equiv (11^3) \cdot (-1 \cdot 13^2) \cdot (-1 \cdot 2^2 \cdot 5^2 \cdot 11) \bmod 5069 \\ (80 \cdot 70 \cdot 63)^2 &\equiv (-1 \cdot 2 \cdot 5 \cdot 11^2 \cdot 13)^2 \bmod 5069 \\ 3039^2 &\equiv (-523)^2 \bmod 5069 \end{aligned}$$

*Da*

$$3039 \not\equiv -523 \bmod 5069$$

*er* $\mathrm{SFD}(3039 - 523, 5069) = 37$ *en faktor.*

I en praktisk implementering af den kvadratiske si, regnes der hele tiden med logaritmen af tallene. Man lagrer således $\log(Q(x))$ i stedet for $Q(x)$, og subtraherer $\log p_i$ i stedet for at dividere $p_i$ de steder, hvor

den er en faktor. Det giver den fordel at man ikke skal regne med så store tal. Samtidig "bytter" man en division ud med en subtraktion. Subtraktionen af $\log p_i$ behøver ikke være 100 % præcis (dvs. man kan nøjes med at regne med enkeltpræcisionstal), da alle tal, der til slut ender på under $\log p_t$, hvor $p_t$ er det største tal i basen, må være $B$-tal. Ellers skulle Q(x) være mindre end $p_t$, og derfor have en faktor mindre end $p_t$. det er der imidlertid checket for, idet primtallene i basen gennemløbes nedefra.

Bemærk at metoden ikke kræver at faktoriseringen af $Q(x)$ opbevares undervejs i sigtningen. Når alle $B$-tallene er fundet, kan de hurtigt faktoriseres, idet man ved at deres primfaktorer er i basen og derfor ikke er større end $p_t$.

### 4.2.4 Afrunding

Den kvadratiske si er implementeret flere steder. De hurtigste implementeringer pt. er foretaget på Sandia National Laboratory, blandt andet for at undersøge sikkerheden af RSA-systemet. I 1984 havde man implementeret metoden på en CRAY X-MP. Denne implementering kunne faktorisere tal på op til 70 cifre [12]. Sidenhen er man gået over til en implementering på 1024 32-bits super-minicomputere, sat sammen til en parallel computer [13]. Denne implementering kører cirka 10 gange hurtigere end CRAY-implementeringen.

# Del II

# Kryptologi

# Kapitel 5

# Edb-sikkerhed

Årsagen til at man anvender kryptografi i edb-systemer, er at man vil øge edb-sikkerheden. Vi vil derfor kort gennemgå begrebet edb-sikkerhed, med henblik på at beskrive hvordan og hvor kryptografi kan være med til at sikre denne. Gennemgangen er baseret på [20].

## 5.1 Fuldstændig edb sikkerhed

Et edb-system er fuldstændig sikret mod trusler, bevidste og ubevidste, hvis man kan arbejde med systemet uden risiko for de følgende typer tab [20] :

### Sikring mod tab af tilgængelighed

Dette indebærer, at det altid skal være muligt at benytte edb-systemets data. Et tab af tilgængelighed kan være totalt. Det betyder at hele edb-systemet er ude af drift, således at ingen af de data, der er i systemet, kan benyttes. Tabet kan også være delvist. Herved menes at man for eksempel ikke kan benytte en enkelt terminal, et pladelager eller en transmissionslinie.

Når tilgængeligheden skal sikres, skal man sørge for at sikre både de bygninger, hvori maskinellet står, selve maskinen, programmer, data, terminaler, strømforsyninger og de linier, der benyttes til transmission, såvel internt som eksternt.

**Sikring mod tab af integritet**

Her sikres at de data, der er i edb-systemet, er korrekte. Det vil sige, at de stemmer overens med virkeligheden, eller med andre ord, at der hverken på grund af bevidste eller ubevidste handlinger er opstået uoverensstemmelse mellem de eksisterende data og den faktiske virkelighed. F.eks. kan måleresultater på grund af støj på transmissionslinien, blive forvansket, så de, når de når frem til computeren, ikke længere er korrekte.

**Sikring mod tab af fortrolighed**

Dette indebærer, at man skal sørge for, at data og programmel kun kan læses og rettes af de rette mennesker. Et eksempel på tab af fortrolighed er hvis en person uretmæssigt får mulighed for at læse indholdet af et fortrolig fil. Derved kan han få fat i følsomme oplysninger om andre personer.

**Sikring mod tab af autenticitet**

Her sikres at de personer man kommunikerer med, også er den de giver sig ud for, og at data man modtager, har den rigtige afsender. Et eksempel er en person, der ved brug af falske bilag hæver penge på en konto, der ikke tilhører ham.

Sikringen skal gælde både den almindelige bruger og programmøren, og både det fysiske system og de data, der findes i systemet.

## 5.2 Trusler mod et edb-system

Skal man opdele de mulige trusler i grupper, kan dette gøres på baggrund af mange forskellige kriterier. Som eksempler kan nævnes; gruppering efter *hvad* truslerne retter sig mod, gruppering efter *hvordan* truslerne kan imødegås, eller gruppering efter *hvordan* de opstår. Vi vil i det følgende først skelne mellem trusler som rammer det fysiske system og trusler som rammer programmel og data. Da de trusler som rammer det fysiske system, ikke kan afhjælpes ved anvendelse af kryptografi, vil vi kun kort ridse nogle af de mulige trusler op. I hver kategori vil vi derudover skelne mellem ubevidste og bevidste trusler.

### 5.2.1 Trusler mod det fysiske edb-system

#### Ubevidste trusler

Dette kan være alt lige fra tekniske uheld til naturkatastrofer. Som et eksempel på en ubevidst trussel kan man nævne en systemoperatør, der falder over strømforsyningen, således at stikket ryger ud, og der opstår strømsvigt. Han kan komme til at knække en diskette, så der ikke længere er adgang til data. Andre mere drastiske eksempler er brand, oversvømmelse, rystelser fra et jordskælv, fugtskader og maskinfejl.

Hvis man vil sikre sit system mod ubevidste trusler, kan man for eksempel fordele sit maskinel i den pågældende virksomhed, så et uheld kun rammer en del af systemet. Det vil så være praktisk, hvis de decentrale maskiner i et stort omfang kan overtage hinandens funktioner. Man må dog gøre sig klart, at en spredning af materiellet gør kommunikationslinierne mere sårbare. En anden mulighed er at have en kopi af alle data på systemet på et lagringsmedie, som opbevares i en boks, der så sikret mod brand, sammenstyrtning og lignende. En tredje mulighed er at have en kopi af hele edb-systemet et andet sted, samt en kopi af data. Flere virksomheder kan eventuelt være fælles om et sådant backup center.

#### Bevidste trusler

De bevidste trusler mod fysiske edb-systemer, kan enten direkte være rettet mod det enkelte system, eller de kan være rettet mod et større område, således at også edb-systemet vil blive ramt. Til disse trusler hører alt fra tyveri af lagringsmedier til krig og sabotage, uautoriseret indtrængen på beskyttede områder eller overvågning og aflytning af lokaler med edb-maskinel.

Skal et system sikres mod disse trusler, kan der dels benyttes de samme midler, som er nævnt ved de ubevidste trusler mod det fysiske system. Derudover kræves supplerende beskyttelsforanstaltninger. Det kan for eksempel være at benytte adgangskontrol, sådan at kun særligt autoriserede personer kan få adgang til maskinellet. En anden mulighed for at sikre bygningen mod uautoriserede indgreb, er ved placering af edb-systemerne under jorden.

#### Opsamling

Konsekvensen af de ovenfor nævnte trusler, vil det primært være tab af tilgængelighed idet systemet ikke fungerer mere, så man ikke kan komme i kontakt med sine data. For de bevidste truslers vedkommende er der yderligere tale om tab af fortrolighed, idet informationer kan komme i de forkerte hænder, hvis for eksempel et magnetbånd stjæles.

Det er ikke muligt at afværge nogle af ovennævnte trusler udelukkende ved at anvende kryptografi. Ved uautoriseret indtrængen i et lokale med edb-maskinel eller ved tyveri af magnetbånd, kan kryptografi dog bruges til at sikre data'ene i det pågældende system.

### 5.2.2 Trusler mod data

#### Ubevidste trusler

De ubevidste trusler medfører at der kommer fejl i de data der benyttes. Det kan være fejl der opstår ved forkert indtastning, eller det kan være programmerings-, syntaks- eller logiske fejl i det program der benyttes. Driftsfejl kan for eksempel opstå hvis to programmer, der forudsætter hinanden, køres i den forkerte rækkefølge. En anden fejl, der kan opstå, er uønsket ændring af data, der er modtaget i forbindelse med en transmission. Et eksempel på en sådan fejl kan være, at der forsvinder et `'ikke'` under transmissionen, så indholdet af data ændres radikalt.

Hvis man skal forbedre datasikkerheden ved at prøve at forhindre sådanne uheld, kan man styre lagring og læsning af data. Dette gøres ofte i operativsystemet, hvor der sørges for at det ikke er muligt at overskrive data, som vedrører en anden fil end den der for øjeblikket arbejdes på. Derudover kan man sikre, at det altid er de korrekte data, der læses fra det pågældende lagringsmedie. En vigtig ting i forbindelse med lagring og læsning af data i databaser, som flere brugere kan benytte uafhængigt af hinanden er, at det ikke på samme tid må være muligt for flere brugere at rette i samme post, da det kan medføre at nogle rettelser går tabt.

Man kan gardere sig mod fejlindtastning i en database ved at benytte et **logretableringssystem**. Dette består af en fil, hvori alle databaseinformationerne registreres med et **førbillede**, som er databasens indhold før en given ændring og et **efterbillede**, som er databasens indhold efter en given ændring. Ved en fejlindtastning, kan man da tilbageskrive

databasens indhold ved hjælp af førbilledet. Ligeledes kan man fremskrive databasens indhold med efterbilledet i tilfælde af program- eller maskinnedbrud.

En vigtig fejlkilde er selve programmerne. De kan være forkerte af flere årsager. Det kan være programmeringsfejl, det kan skyldes misforståelser på grund af for dårlig kommunikation om, hvad programmet egentlig skal kunne, eller det kan være noget så simpelt som en indtastningsfejl.

Man kan sikre sig mod fejl som opstår ved transmission af data fra et sted til et andet, ved at tilføje redundante bit til data, såkaldte **fejldetekterende** eller **fejlkorrigerende koder**. En fejldetekterende kode kan vise om noget er transmitteret galt. Det kan for eksempel ske med checksummer, der viser om antallet af ettaller og nuller er rigtigt. En fejldetekterende kode kan ikke vise hvad der er galt, så opdages der en fejl, må bitstrengen transmitteres igen. En fejlkorrigerende kode kan også vise hvor fejlen er, så den kan rettes.

### Bevidste trusler

Dette er trusler, der er resultat af bevidste menneskelige handlinger, hvor formålet er at forringe datasikkerheden.

En bevidst trussel kan være at indholdet i en fil ændres, så gælden på en bankkonto vendes til et tilgodehavende, at en fil slettes så oplysninger forsvinder, eller at fortrolige data kopieres, så eksamensopgaver falder i hænderne på en studerende inden eksamen. Dette kan blandt andet udføres ved brug af skjulte tilføjelser i edb-systemet. Tapning af information fra en transmissionslinie kan for eksempel ske ved hjælp af specielle **lytteprogrammer**, der ligger skjult i systemet.

Andre muligheder for at udføre bevidste trusler er ved læsning af brugte, men ikke slettede lagerområder, eller opsamling af papiraffald indeholdende fortrolige oplysninger. En sidste mulighed er fremstilling af en model for et område af virkeligheden. En sådan model kan benyttes til at planlægge og kontrollere udviklingen på det pågældende område. Som eksempel kan nævnes en virksomhedsindehaver, der simulerer sin virksomheds totale regnskab. Ved køre modellen baglæns kan han fremstille falske bilag, på baggrund af hvilke, han kan opnå store lån.

Skal man sikre sig mod disse angreb, kan man dels benytte metoderne nævnt under de ubevidste trusler. Derudover kan der anvendes kryptografi. Man kan kryptere de meddelelser, der skal sendes, således at

det ikke er muligt at læse og forstå indholdet af meddelelserne. Dette medfører at det kun er modtageren som kender koden, der ved hvordan meddelelsen kan afkodes og læses. Hvordan dette foregår er det centrale emne i dette projekt. Kryptografi kan også anvendes til at sikre meddelelsernes autenticitet og deres integritet. Kun den der kender koden, kan jo have afsendt meddelelsen.

Skal man sikre sig mod, at ikke hvem som helst har mulighed for få adgang til edb-systemet, kan der benyttes **passwords**. I den forbindelse kan kryptografi også anvendes, da det er vigtigt, at man ikke uden videre kan læse passwords, der beskytter mod uautoriseret indtrængen i et system.

Data bør beskyttes både i den tid de anvendes, og også efter at de er blevet slettet. F.eks. kan filer, der er markeret som slettede ofte stadig læses, da de pågældende lagerområder sjældent bliver overskrevet med det samme. Selv om dataområderne er blevet overskrevet, er det ofte muligt at læse hvad der tidligere har stået, flere generationer bagud.

### Opsamling

Konsekvensen af en forringelse af datasikkerheden, hvad enten det drejer sig om bevidste eller ubevidste trusler, er tab af tilgængelighed og tab af integritet. Tilgængeligheden tabes når man overskriver eller sletter vigtige data, mens integriteten tabes hvis man ændrer data, så de ikke svarer til den faktiske virkelighed. Ved de bevidste trusler mod datasikkerheden sker der ligeledes et tab af fortroligheden, for eksempel ved en opsamling af papiraffald med henblik på læsning af beskyttede data.

## 5.3 Hvornår skal der foretages sikkerhedsforanstaltninger

I den virkelige verden vil man ikke nødvendigvis foretage alle de her nævnte sikkerhedsforanstaltninger i et edb-system. Her i landet vil det for eksempel være dumt at sikre sig mod jordskælv, da risikoen for sådanne er forsvindende lille.

Når man arbejder med at finde ud af, hvornår man skal indføre en given sikkerhedsforanstaltning benytter man **risikostyring**. Herved

forstås en systematisk indføring af sikkerhedsforanstaltninger med det formål, at omkostningerne ved en given sikkerhedsforanstaltning skal være mindre end det forventede tab, som den givne sikkerhedsforanstaltning vil forhindre eller afbøde. Når nye sikkerhedsforanstaltninger skal indføres, er det vigtigt, at de analyseres grundigt, da værdien af dem er meget afhængig af den geografiske placering, hvem man er og hvilke sikkerhedsforanstaltninger, der i forvejen er installeret.

Da der kan investeres ubegrænsede ressourcer i sikkerhedsforanstaltninger, er det vigtigt at overveje, hvornår det punkt vil komme, hvor omkostningerne til yderligere sikkerhed overstiger det tab, som sikkerhedsinvesteringen skal forebygge. Derfor skal det pågældende edb-system nøje vurderes. Hvor vigtige er de data, der er lagret og hvor meget vil det koste at reetablere tabte data i forhold til prisen for at indføre de givne sikkerhedsforanstaltninger ? Derudover vil man ofte være villig til at betale lidt ekstra for at undgå de helt store katastrofer, ikke mindst fordi man sjældent vil være i stand til at starte helt forfra med at genopbygge et edb-system.

I forbindelse med anvendelse af kryptografiske systemer, kan man afhængigt af den ønskede sikkerhedsgrad benytte forskellige kombinationer af kryptografiske systemer. Det er således en dårlig ide at anvende et meget raffineret, tungt og dyrt system, hvis de data, der skal beskyttes, ikke repræsenterer nogen reel værdi. Omvendt skal man ikke sikre nationalbanken med en billig letbrydelig kode.

### 5.3.1 Opsamling

Den ovenstående gennemgang viste at kryptografi kunne anvendes til at sikre mod bevidste trusler mod datasikkerheden, specielt trusler mod fortrolighed, integritet og autenticitet. Det er derimod mindre anvendeligt til trusler mod det fysiske system, og samt til ubevidste trusler mod datasikkerheden. Vi vil senere i projektet uddybe anvendelsen af kryptografi i edb-systemer.

Fejldetekterende og fejlkorrigerende koder som vi nævnte ovenfor, er et emne der er tæt relateret til kryptografi. Begge dele benytter indkodning, men til hver sit formål. Hvor kryptografi beskytter mod de bevidste trusler, beskytter de fejlkorrigerende koder mod de ubevidste trusler mod datasikkerheden.

# Kapitel 6

# Kryptologi

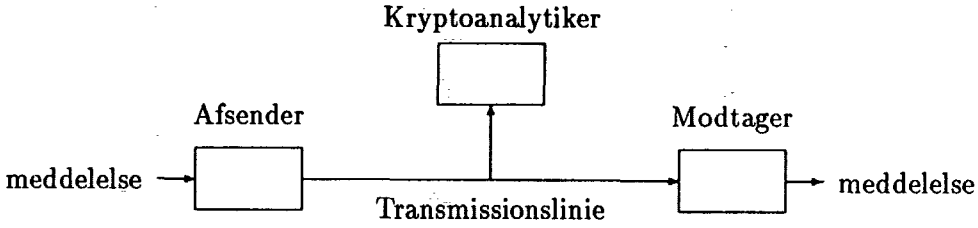
Efter den generelle beskrivelse af datasikkerhed, vil vi nu gå i dybden med kryptografi. Kapitlets tre første afsnit beskriver de grundlæggende begreber i et kryptografisk system. Derefter vil vi gennemgå nogle forskellige kryptografiske teknikker. Vi afslutter kapitlet med en diskussion af sikkerhed i kryptografiske systemer.

## 6.1 Grundlæggende begreber

I indledningen definerede vi kryptologi som *"læren om at skjule"*. Kapitel 5 om datasikkerhed viste at kryptografi ikke blot kan bruges til at holde ting hemmelige, men også til at sikre data's autenticitet og integritet. Med andre ord kan man definere kryptografi, som en metode til at forhindre bevidste men for brugeren utilsigtede hændelser i kommunikation.

Der skelnes normalt mellem tre forskellige begreber :

- **Kryptografi** som er at udvikle kryptografiske systemer, og at kryptografere meddelelser.
- **Kryptoanalyse** som er at prøve på at bryde fortroligheden i et kryptografisk system.
- **Kryptologi** som er studiet af kryptografi og kryptoanalyse.




**Figur 6.1:** Et kryptografisk system

Kryptologi er altså et overordnet begreb i forhold til de to andre. De personer der arbejder med henholdsvis kryptografi, kryptoanalyse og kryptologi, kaldes henholdsvis **kryptografer**, **kryptoanalytikere** og **kryptologer**. Man kan således sige at kryptografer og kryptoanalytikere søger at gøre livet surt for hinanden, mens kryptologerne interesseret ser på.

I figur 6.1 ses en skematisk tegning af et **kryptografisk system**, også kaldet et kryptosystem. Det består af en **afsender**, som sender en meddelelse til en **modtager** over en ikke-fortrolig kommunikationslinie. På denne linie sidder der en kryptoanalytiker, som forsøger at opfange og/eller ændre meddelelsen. Bemærk at meddelelsen ikke behøver at være tekst. Der kan ligeså godt være tale om billeder eller lyd. På samme måde kan kommunikationslinien være alt fra radiobølger og kabler til et brev, bragt frem af en kurer til hest.

### 6.1.1 Den grundlæggende kryptografiske teknik

Den grundlæggende kryptografiske metode bygger på at tage den originale meddelelse – **klarteksten** – og transformere den, således at der fremkommer en ny tekst kaldet **cifferteksten** eller **kryptogrammet**. Denne proces kaldes **indkodning**. Cifferteksten kan kun læses, hvis man ved, hvordan man transformerer den tilbage til klartekstform. Dette kaldes **afkodning**. Fidusen er så, at både afsender og modtager kender de transformationer der henholdsvis ind- og afkoder teksten. Disse transformationer skal holdes hemmelige, da det kun er afsenderen og modtageren, der må kunne læse den pågældende meddelelse.

Kryptoanalytikerens primære mål er at genskabe hele klarteksten eller dele deraf. Dette kan enten gøres ved at gætte transformationen eller på anden måde at fravriste cifferteksten noget af dens oprindelige information. Hvis kryptoanalytikeren får genskabt hele klarteksten, siges hun at have brudt koden, eller at have afkodet cifferteksten.

Sædvanligvis lader man de transformationer der ind- og afkoder henholdsvis klar- og ciffertekst være styret af en **indkodningsalgoritme** og en **afkodningsalgoritme** samt en **indkodningsnøgle** og **afkodningsnøgle**. Ved at lade transformationen være styret af både en algoritme og en nøgle opnår man en række fordele.

For det første kan man nøjes med at hemmeligholde nøglen, og lade transformations algoritmen være offentlig kendt. Dette gør at sikkerhedsforanstaltningerne bliver enklere, da det er nemmere at hemmeligholde en relativt kort nøgle end en lang algoritme. Det har også den fordel, at algoritmen kan udveksles mellem afsender og modtager i fuld offentlighed.

For det andet kan man, hvis koden bliver brudt, ofte nøjes med at udskifte selve nøglen, hvad der er langt enklere end at udskifte algoritmen, som typisk vil være en del af et større program.

### *Eksempel 6.1. Cæsars ciffer*

*Det simpleste og måske mest kendte eksempel på et kryptografisk system er den såkaldte 'Cæsars ciffer', som er opkaldt efter Julius Cæsar. Her foregår indkodningen ved, at man skifter hvert enkelt bogstav i klarteksten ud med det bogstav, der står et bestemt antal pladser til højre for klartekst bogstavet i alfabetet.*

*Vælger man, som Cæsar gjorde, at skifte klartekstbogstaverne 3 pladser til højre, bliver* `A` *til* `D`, `B` *til* `E`, ..., `Å` *til* `B` *og mellemrum til* `C`. `ABE` *indkodes således til* `DEH`.

*Afkodningen foregår helt analogt, ved enten at skifte alle de enkelte bogstaver 3 pladser til venstre eller ved at skifte*

$$\text{“antal bogstaver i alfabetet”} - 3 = 30 - 3 = 27$$

*pladser til højre.*

*Hvis vi generelt beskriver indkodningsalgoritmen som*

"Skift $x$ antal pladser til højre",

*bliver indkodningsnøglen $x$. Man kan så enten lade afkodningsalgoritmen være lig indkodningsalgoritmen, og bruge $y = 30 - x$ som afkodningsnøgle, eller man kan bibeholde indkodningsnøglen, og som afkodningsalgoritme bruge*

"Skift $x$ antal pladser til venstre"

*Den enkleste metode til at bryde dette system er ved at afprøve alle 30 nøgler, indtil klarteksten kommer frem. Denne metode til at bryde et system kaldes for* **udtømning af nøglerummet**. *Dette viser nødvendigheden af at antallet af mulige nøgler i et kryptosystem, skal være så stort, at en systematisk afprøvning af alle nøgler ikke er mulig indenfor en rimelig tid.*

### 6.1.2 En model af et kryptosystem

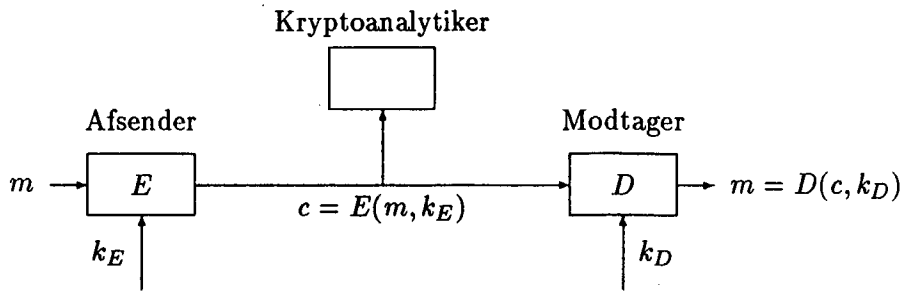
Alle kryptosystemer lader sig beskrive af en generel model, der består af følgende :

- En mængde af mulige klartekstmeddelelser $M$.
- En mængde af mulige ciffertekstmeddelelser $C$.
- En indkodningsnøgle $k_E$.
- En afkodningsnøgle $k_D$.
- En indkodningsalgoritme[1] $E$.
- En afkodningsalgoritme $D$.

Man indkoder en meddelelse $m \in M$, ved at udføre algoritmen $E$ på meddelelsen $m$ med indkodningsnøglen $k_E$. Ciffertekstmeddelelsen $c$ er derfor givet ved :

$$c = E(m, k_E) \tag{6.1}$$

[1] E for det engelske Encode, og D for Decode.




**Figur 6.2:** En model af et kryptografisk system

Afkodningen foregår tilsvarende med algoritmen $D$ på cifferteksten $c \in C$ og med afkodningsnøglen $k_D$ :

$$m = D(c, k_D) \tag{6.2}$$

Det gælder følgelig at

$$m = D(E(m, k_E), k_D) \tag{6.3}$$

Denne model er vist i skematisk form på figur 6.2.

### *Eksempel 6.2. Cæsars ciffer fortsat*

*Med meddelelsen* `MISSISSIPPI`*, vil de nævnte begreber have følgende værdier, når henholdsvis ind- og afkodningsalgoritmen er Cæsars ciffer :*

| | | | | |
|---|---|---|---|---|
| $m$ | = | | | `MISSISSIPPI` |
| $M$ | = | | | *Alle tekststrenge* |
| $C$ | = | | | *Alle tekststrenge* |
| $k_E$ | = | | | 3 |
| $E$ | = | | | *"Skift $k_E$ pladser til højre."* |
| $c$ | = | $E(m, k_E)$ | = | `PLVVLVVLSSL` |
| $D$ | = | | | $E$ |
| $k_D$ | = | | = | $30 - 3 = 27$ |
| $m$ | = | $D(c, k_D)$ | | `MISSISSIPPI` |

Bemærk at i ovenstående eksempel bliver alle `I`'er indkodet til `L`'er, alle `S`'er til `V`'er og alle `P`'er til `S`'er.

## 6.2 Opdeling af kryptosystemer

Kryptosystemer opdeles helt overordnet efter følgende kategorier [5] :

- *Begrænsede kryptosystemer.*
  Systemer uden nøgler.
- *Generelle kryptosystemer.*
  Systemer med nøgler.
  - *Hemmelig-nøgle (symmetriske) kryptosystemer.*
    Systemer hvor $k_E = k_D$.
  - *Offentlig-nøgle (asymmetriske) kryptosystemer.*
    Systemer hvor $k_E \neq k_D$.

### 6.2.1 Begrænsede kryptosystemer

Et begrænset kryptosystem er et system uden nøgler. Det betyder at ind- og afkodningsalgoritmerne $E$ og $D$, kun får henholdsvis klar- og cifferteksten som parametre :

$$c = E(m) \text{ og } m = D(c)$$

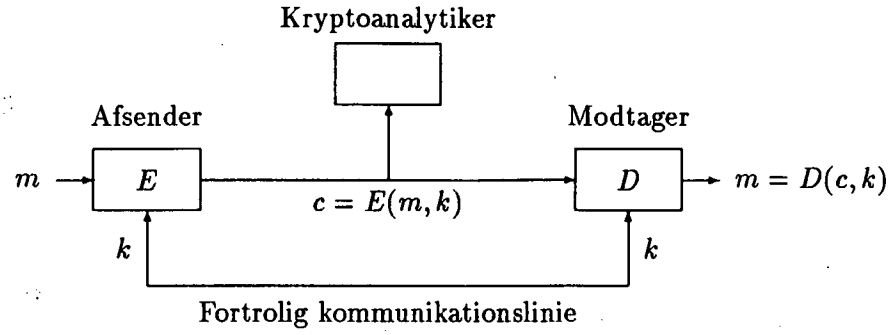
Et simpelt eksempel på et begrænset kryptosystem er et system, hvor man indkoder en meddelelse ved at skrive den bagfra. F.eks. indkodes `MISSISSIPPI` som `IPPISSISSIM`. Dette er et eksempel på et kryptosystem, hvor ind- og afkodningsalgoritmerne er ens, så $E = D$. Dette behøver nødvendigvis ikke at være tilfældet for et begrænset system. Der skal gælde følgende for et begrænset kryptosystem :

- $E$ og $D$ skal være hinandens inverse :

$$E = D^{-1}$$

- $E$ og $D$ skal være lette at beregne.

Som nævnt i afsnit 6.1.1 foretrækkes normalt de mere fleksible kryptosystemer med nøgler. Af samme grund vil der heller ikke blive gjort mere ud af disse begrænsede kryptosystemer i denne rapport.




**Figur 6.3:** Et Hemmeligt-nøgle kryptografisk system

### 6.2.2 Generelle kryptosystemer

Hvis sikkerheden af et kryptosystem ikke udelukkende er afhængig af ind– og afkodningsalgoritmerne men også af en eller flere nøgler, kaldes kryptosystemet for generelt. Et eksempel på et sådant kryptosystem er Cæsars ciffer, fordi nøglen kan udskiftes.

De generelle systemer deler sig videre i to undergrupper : Hemmelig-nøgle kryptosystemerne og offentlig-nøgle kryptosystemerne.

#### Hemmelig-nøgle systemer

Et system hvor ind- og afkodningsnøglerne er identiske, eller hvor den ene let kan beregnes ud fra den anden, kaldes et hemmelig-nøgle kryptosystem eller et symmetrisk kryptosystem. I et hemmelig-nøgle system må afsender og modtager enes om en fælles hemmelig nøgle, som skal transmitteres mellem de implicerede parter, inden den fortrolige kommunikation kan starte. Denne transmission skal nødvendigvis foregå over en sikker kanal, da en kryptoanalytiker ellers kan opsnappe nøglen og dermed aflæse hele den efterfølgende kommunikation. Denne type kryptosystem, er i skematisk form vist på figur 6.3.

Et hemmelig-nøgle system skal opfylde følgende betingelser :

- $k_E = k_D$ eller $k_E = T(k_D)$, hvor $T$ er en let beregnelig algoritme.

- For enhver nøgle $k$ skal $E$ og $D$ være hinandens inverse, dvs at :

$$\forall k \; : \quad D(k, E(k, m)) = m$$

- $E(k, m)$ og $D(k, c)$ skal være lette at beregne.

- Ind- og afkodningen af meddelelserne må ikke kunne gennemføres uden brug af henholdsvis ind- og afkodningsnøglerne.

Det mest udbredte hemmelignøglesystem er DES (Data Encryption Standard), som blev udviklet af IBM i samarbejde med National Bureau of Standards (NBS) i USA, og offentliggjort i 1977, se f.eks. [38]. Algoritmen bygger på at meddelelserne ind- og afkodes i blokke på 64 bits ved brug af en nøgle på 56 bits. Først underkastes de enkelte klartekstblokke en permutation, hvor bittene blandes. Herefter foretages der ved hjælp af forskellige nøgler, som alle er afledt af den oprindelige 56 bit nøgle, 16 efterfølgende indkodninger af den oprindelige klartekst. Efter de 16 indkodninger permuteres klartekstblokken med den inverse permutation til den, der startede indkodningen. Dette er vigtigt for at sikre, at afkodningen kan foretages ved at gennemløbe de 16 indkodninger i omvendt rækkefølge.

Hemmelig-nøgle kryptosystemer lider af en svaghed, som kan være kritisk, hvis de skal bruges i et større edb-netværk. Antag at man kobler 1000 terminaler sammen i et netværk, og ønsker at give dem mulighed for at kommunikere fortroligt sammen to og to. Der skal så med et hemmelig-nøgle kryptosystem bruges

$$n \cdot (n-1)/2 = 1000 \cdot 999/2 = 499.500$$

nøgler, idet hver af de 1000 enheder i netværket, skal dele en nøgle med hver af de 999 andre. Ydermere skal disse knap 500.000 nøgler først sendes rundt via fortrolige kanaler, inden den kryptograferede kommunikation kan starte. Endelig vil tilkomsten af bare 1 ny enhed på netværket, medføre at der skal genereres 1000 nye nøgler, der skal distribueres fortroligt.

Dette førte til udviklingen af offentlig-nøgle kryptosystemerne.

### Offentlig-nøgle kryptosystemer

Et offentlig-nøgle kryptosystem eller et **asymmetrisk system**, er et system hvor ind- og afkodningsnøglerne $k_E$ og $k_D$ er forskellige. Dette skal forstås således, at den ene nøgle ikke let må kunne beregnes ud fra den anden. Kryptosystemet kaldes offentlig-nøgle system fordi princippet er, at hver bruger genererer et sæt ind- og afkodningsnøgler. Indkodningsnøglen $k_E$ offentliggøres samtidig med, at afkodningsnøglen $k_D$ holdes hemmelig.

I et sådant system vil man frit kunne offentliggøre indkodningsnøglen, da kryptoanalytikeren alligevel ikke kan bruge denne viden til noget. Herved opnår man, at behovet for forudgående kommunikation over en sikker kanal bortfalder, og at problemet med de mange nøgler reduceres drastisk. Alle der ønsker at sende meddelelser til en bestemt person, kan bruge den samme indkodningsnøgle. Derfor er der kun brug for et sæt nøgler pr bruger i et netværk. I eksemplet fra før, mindskes antallet af nøgler således fra knap $\frac{1}{2}$-million til 1000 og hver enkelt udvidelse kræver kun *et* nyt sæt nøgler.

Systemet skal have følgende egenskaber :

- Indkodning og afkodning er hinandens inverse.

$$D(k_D, E(k_E, m)) = m$$

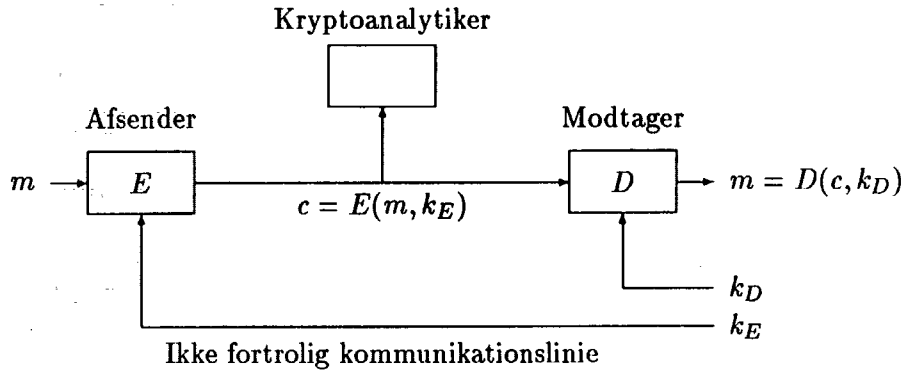
- Det skal være meget vanskeligt, typisk hurtigst i løbet af nogle år, at afkode ciffertekster uden den hemmelige nøgle. $k_E$ må således *ikke* bruges til at afkode :

$$D(k_E, E(k_E, m)) \neq m$$

  Endvidere må kendskabet til $k_E$ ikke give nogen information om, hvordan $k_D$ skal beregnes. Med andre ord må der ikke findes en let beregnelig algoritme $T$ således at :

$$T(k_E) = k_D$$

- Det skal være let at generere korresponderende $k_E$ og $k_D$ par.
- Både $D(k_D, c)$ og $E(k_E, m)$ skal være lette at beregne.




**Figur 6.4:** Et offentlig-nøgle kryptografisk system

I diagramform ser denne type kryptosystem ud som på figur 6.4.

I et offentlig-nøgle kryptosystem er transmissionsproceduren altså som følger :

En afsender Anne som vi kalder A, skal sende en fortrolig meddelelse til en modtager Bjarne, som vi kalder B.

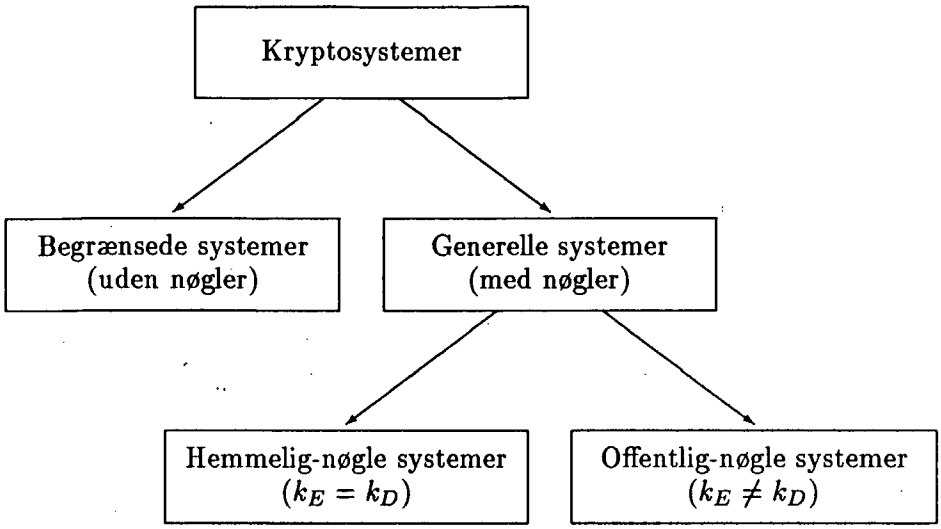
1. B genererer en indkodningsnøgle $k_E$ samt den korresponderende afkodningsnøgle $k_D$. Han skal holde $k_D$ hemmelig.

2. B sender derefter $k_E$ til A. Dette behøver ikke at foregå over en hemmelig linie, da $k_E$ må være offentligt kendt.

3. A indkoder sin meddelelse $m$ med algoritmen $E$ og nøglen $k_E$. Hun sender den beregnede ciffertekst $c = E(k_E, m)$ til B.

4. B bruger sin hemmelige nøgle $k_D$ til at afkode $c$, hvorved han kan læse meddelelsen $m = D(k_D, c)$.

Denne procedure kan udvides således, at B kan modtage fortrolige meddelelser fra flere personer. Dette gøres ved at den offentlige nøgle $k_E$ kan placeres i en slags telefonbog over indkodningsnøgler. Enhver, der har noget på hjerte, kan herefter indkode en meddelelse ved hjælp af denne nøgle. Afsenderen har, når først meddelelsen er indkodet, ingen mulighed for at læse sin egen ciffertekstmeddelelse, bortset fra ved at forsøge bryde koden.




**Figur 6.5:** En opdeling af kryptografiske systemer

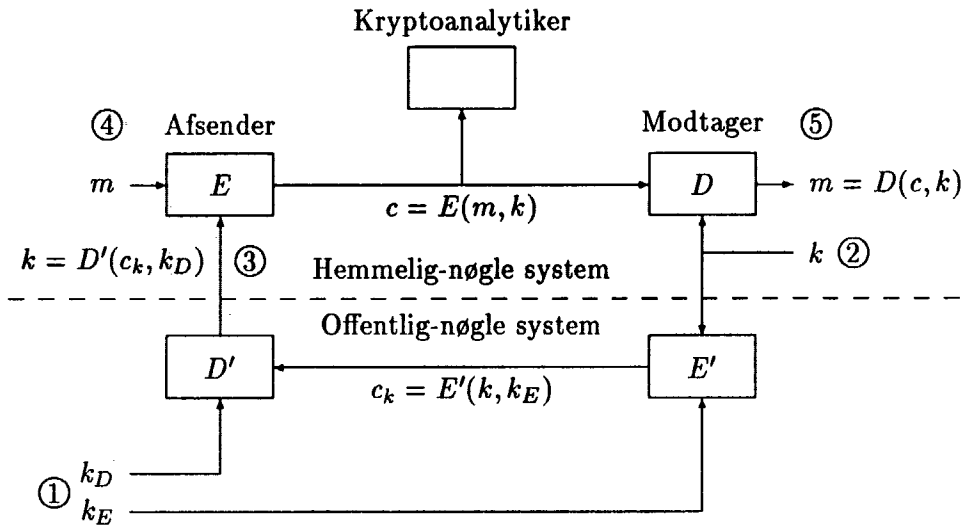
Ideen til et offentlig-nøgle system blev først fremsat af Diffie og Hellman[17] i 1976. Dette skete dog uden, at de fremkom med et konkret forslag til, hvordan et sådant system kunne konstrueres. Det er der derimod flere andre, der siden har gjort. Det mest kendte er RSA-kryptosystemet, som gennemgås i kapitel 8.

### 6.2.3 Opsamling

Figur 6.5 giver et overblik over de forskellige typer kryptosystemer, som vi netop har beskrevet.

Til kommercielt brug er det i dag hemmelig-nøgle systemerne, der er fremherskende. Dette skyldes primært at de indkodningshastigheder, der kan opnås i dag, er væsentlig højere for de bedste hemmelig-nøgle systemer end for offentlig-nøgle systemerne.

Hvis man ønsker at kombinere hemmelig-nøgle kryptosystemernes høje indkodningshastigheder med offentlig-nøgle kryptosystemernes etablering uden forudgående kommunikation, kan man benytte et såkaldt




**Figur 6.6:** Et hybrid-system

**hybrid-system.** Et hybrid-system er et system, hvor man både benytter et offentlig- og et hemmelig-nøgle kryptosystem. Hemmelig-nøgle systemet benyttes til transmission af selve meddelelserne, mens offentlig-nøgle systemet benyttes til distribution af hemmelig-nøgle systemets nøgler.

Hvis vi antager at modtageren har den hemmelige $k$, bliver transmissionsprotokollen i et hybridsystem (figur 6.6) som følger :

1. Afsenderen genererer et sæt nøgler til et offentlig-nøgle system, og sender den offentlige nøgle til modtageren.
2. Derefter indkoder modtageren den hemmelige nøgle $k$ med offentlig-nøgle systemet, og sender den til afsenderen.
3. Afsenderen afkoder $k$ med den hemmelige nøgle $k_d$ fra offentlig-nøgle systemet.
4. Afsenderen kan nu indkode sin meddelelse med $k$ og sende den til modtageren.
5. Modtageren kan endelig afkode meddelelsen med $k$.

Er det afsenderen der har den hemmelige nøgle, er det modtageren der skal generere nøglesættet.

Et eksempel på et hybridsystem er dankortautomaterne, se afsnit 12.2.

## 6.3 Kryptoanalyse

Kryptoanalyse er som tidligere nævnt, det at forsøge at bryde fortroligheden i et kryptografisk system. Med andre ord vil kryptoanalytikeren dels prøve at genskabe klarteksten svarende til en bestemt ciffertekst og dels (og helst) prøve at finde den benyttede indkodningsnøgle.

Inden for hemmelignøgle krypto-analysen skelner man som regel mellem tre forskellige angreb afhængigt af, hvor meget information kryptoanalytikeren har til sin rådighed.

- **Rent ciffertekst angreb.**
  I et rent ciffertekst angreb, har kryptoanalytikeren kun nogle stumper ciffertekst til sin rådighed og ingen klartekst. Hun skal så enten finde nøglen ud fra cifferteksten eller, hvis det ikke er muligt, genskabe så meget af klarteksten svarende til cifferteksterne som muligt.

- **Kendt klartekst angreb.**
  Her har kryptoanalytikeren både noget ciffertekst og de dertil svarende klartekster til rådighed. Igen skal hun enten finde nøglen, hvis det er muligt, eller prøve at afkode nogle nye cifferteksterne.

- **Valgt klartekst angreb.**
  I et valgt klartekst angreb får kryptoanalytikeren selv lov til at vælge nogle klartekster, og får så de dertil hørende cifferteksterne. Opgaven er så som før at finde nøglen på baggrund af de korresponderende klar- og cifferteksterne.

For bedst muligt at kunne demonstrere forskellene mellem de tre typer angreb, vil vi som eksempel benytte en kode, der bruger simpel bogstav substitution. I denne kode knytter man til hvert bogstav i alfabetet et andet bogstav, som er det bogstav det oprindelige bogstav skal udskiftes med, når man indkoder teksten. Et eksempel er vist i tabel 6.1. Her bliver `ABE` til `QWT` osv.

| Klartekst | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Æ | Ø | Å |
| Q | W | E | R | T | Y | U | I | O | P | Å | A | S | D | F | G | H | J | K | L | Æ | Ø | Z | X | C | V | B | N | M |

Ciffertekst

**Tabel 6.1:** Simpel bogstav substitution

I et rent ciffertekst angreb vil det som regel være ret besværligt at bryde den ovennævnte kode, men det er dog muligt ved frekvensanalyse. Det er velkendt, at de forskellige bogstaver i alfabetet ikke alle optræder lige hyppigt i en normal tekst. F.eks. er *e, n, r, s* og *t* blandt de hyppigste bogstaver i det danske sprog, mens *q, w, z* og *å* er mere sjældne. Kan man finde tilsvarende svingninger i hyppigheden af bogstaverne i cifferteksten, kan man som regel gætte sig til, hvilke bogstaver i cifferteksten, der svarer til bestemte bogstaver i klarteksten. Giver dette ikke noget resultat, kan man på tilsvarende måde forsøge sig med hyppigt forekommende bogstavpar f.eks. *sk, hv* og *ll*, eller bogstavtripler f.eks. *lle, mme, ede* og *der*.

I et kendt klartekst angreb vil den ovenstående kode nemt kunne brydes, idet man umiddelbart får koden for de bogstaver som er nævnt i teksten. Hvis man har en klar- eller ciffertekst indeholdende alle bogstaver i alfabetet er koden brudt.

I et valgt klartekst angreb er eksempelkoden umiddelbart brudt, idet man blot vælger klarteksten `ABCDEFGIHJK....Å`. Heraf kan man direkte aflæse den benyttede nøgle.

For offentlig-nøgle systemer vil de tre ovenstående kategorier falde sammen til et valgt-klartekst angreb, idet man her selv kan indkode vilkårligt meget klartekst. Derudover opstår der en ny kategori :

- **Valgt ciffertekst angreb.**
  Her vælger kryptoanalytikeren nogle ciffertekster og får adgang til de tilsvarende klartekster. Hendes opgave er som før enten at finde den anvendte nøgle eller at få afkodet nogle nye ciffertekster.

## 6.4 Kryptografiske teknikker

Efter i de foregående afsnit at have beskrevet, hvordan de forskellige kryptosystemer er opbygget, vil vi nu beskrive hvilke teknikker, der bruges til konstruktion af disse kryptosystemer.

### 6.4.1 Blok– og strømkryptering

Grundlæggende er der to måder, hvorpå man kan indkode en klartekstmeddelelse. Det er blok– og strømkryptering.

#### Blokkryptering

I blokkryptering splittes en meddelelse $m$ op i lige store dele $m_1, m_2, \ldots$, som indkodes hver for sig med nøglen $k_E$ :

$$c = E(m_1, k_E), E(m_2, k_E), \ldots$$

Der benyttes altså den samme nøgle og den samme algoritme på alle blokkene.

Cæsars ciffer er et eksempel på en kode, der benytter blokkryptering. Blokkene har da længden 1 bogstav, idet der benyttes den samme nøgle til alle bogstaver.

I blokkryptering er der imidlertid det problem, at identiske tekstblokke altid indkodes til den samme stump ciffertekst. Det betyder, at en meddelelses struktur ikke sløres, når den indkodes. Dette er specielt kritisk, hvis man indkoder meget strukturerede meddelelser som f.eks. edb-programmer[2]. Her vil programmets struktur træde tydeligt frem, selv når det er indkodet. F.eks. vil alle `BEGIN/END` blokke tydeligt kunne ses, idet hvert `BEGIN` og hvert `END` altid indkodes til den samme ciffertekst. Ligeledes vil de indryk, man normalt laver i edb programmer, kunne afsløres.

[2]Hermed ikke sagt at alle skriver strukturerede edb-programmer. Denne diskussion bør i hvert fald ikke tages som et argument *for* spaghetti-programmering.

### Strømkryptering

I strømkryptering indkoder man meddelelsen bit for bit eller bogstav for bogstav, og for hver del af meddelelsen tages en ny del af nøglen :

$$c = E(m_1, k_{E_1}), E(m_2, k_{E_2}) \ldots$$

I dette tilfælde kan man risikere, at nøglen ikke er lang nok. Er dette tilfældet, må man tage blokkryptering i brug, idet man på et tidspunkt begynder forfra på nøglen :

$$c = E(m_1, k_{E_1}), E(m_2, k_{E_2}) \ldots E(m_n, k_{E_n}), E(m_{n+1}, k_{E_1}) \ldots$$

Dette kaldes **periodisk strømkryptering.**

Strømkrypteringen kan også foregå ved at nøglen $k_E$ benyttes som startværdi i en algoritme, der generer nye nøgler for hver ny del af meddelelsen :

$$c = E(m_1, k_E), E(m_2, f(k_E)), E(m_3, f(f(k_E))), \ldots$$

Man kan udbygge Cæsars ciffer, således at det første bogstav rykkes $k_1$ pladser til højre, det andet bogstav $k_2$ pladser, ..., det $n$'te bogstav $k_n$ pladser og det $n+1$'te bogstav rykkes igen $k_1$ pladser til højre. Dette er et eksempel på et system, der benytter strømkryptering.

Hvis vi eksempelvis vælger $n = 3$, og

$$k_1 = 3, \quad k_2 = 5 \text{ og } k_3 = 20$$

bliver `MISSISSIPPI` indkodet som vist i tabel 6.2. `MISSISSIPPI` bliver altså til `PNIVNIVNFSN`. Bemærk at `S` både indkodes som `I` og `V`.

I strømkryptering kan man altså ikke regne med at identiske ciffertekstbogstaver korresponderer med identiske klartekstbogstaver. Forskellige dele af teksten indkodes ved strømkryptering med forskellige dele af nøglen, som kan være af variabel længde afhængig af de pågældende sikkerhedskrav. Strømkryptering lider derimod af en anden svaghed, idet den er følsom overfor, om der forsvinder en del af teksten under transmissionen. Sker dette, vil hele afkodningen gå i vasken, da ciffertekststrømmen og nøglestrømmen *'kommer ud af takt'*.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| M | er bogstav nr. | 13 | → | 13+ 3 | = | 16 | som er | P |
| I | " | 9 | → | 9+ 5 | = | 14 | " | N |
| S | " | 19 | → | 19+20 | = | 9 | " | I |
| S | " | 19 | → | 19+ 3 | = | 22 | " | V |
| I | " | 9 | → | 9+ 5 | = | 14 | " | N |
| S | " | 19 | → | 19+20 | = | 9 | " | I |
| S | " | 19 | → | 19+ 3 | = | 22 | " | V |
| I | " | 9 | → | 9+ 5 | = | 14 | " | N |
| P | " | 16 | → | 16+20 | = | 6 | " | F |
| P | " | 16 | → | 16+ 3 | = | 19 | " | S |
| I | " | 9 | → | 9+ 5 | = | 14 | " | N |

**Tabel 6.2:** Strømkryptering

### CiffertekstBlokKobling – CBK

Denne teknik kan benyttes til at imødegå svaghederne nævnt under blok- og strømkryptering kan man benytte denne teknik. I stedet for at indkode blokkene uafhængigt at hinanden som i blok– og strømkryptering, kan man lade enhver ciffertekstblok afhænge af den tidligere klartekstblok. En fordel ved denne metode er, at en kryptoanalytiker ikke kan bruge løsrevne stumper til at forvirre modtageren. Desuden er CBK[3] selvsynkroniserende, så det kun er et begrænset antal blokke der vil gå tabt, i tilfælde af transmissionsfejl.

CBK foregår ved at beregne den $i$'te ciffertekstblok $c_i$ som

$$c_i = E(X(m_i, c_{i-1}), k_E)$$

hvor funktionen $X(a, b)$ er en funktion der blander de to blokke. Det kan f.eks. være den logiske funktion $a$ XOR $b$. For at afkode $c$ skal modtageren udover sin private afkodningsnøgle kende $c_0$, der er en kunstig ciffertekstblok, som anvendes ved indkodning af den første blok. Modtageren beregner

$$m_i = X(D(c_i, k_D), c_{i-1})$$

En transmissionsfejl vil kun ødelægge to blokke, idet hver blok kun afhænger af $c_i$ og $c_{i-1}$.

[3]På engelsk Cipher Blok Chaining – CBC

## 6.5 Sikkerhed af det kryptografiske system

Sikkerheden af et kryptografisk system er et mål for, hvor let eller svært det er bryde den pågældende kode. Overordnet set er der to forskellige mål for, hvor god sikkerheden i et kryptografisk system er : Den **teoretiske** og den **beregnelige sikkerhed** [16].

Begreberne teoretisk og beregnelig sikkerhed blev defineret af Shannon efter anden verdenskrig. Der skabte han grundlaget for den videnskabelige eller den matematiske kryptologi. Han var den første, der på en formaliseret måde beskrev kryptografiske systemer og opstillede sikkerhedsmål for dem.

### 6.5.1 Teoretisk sikkerhed

Dette beskriver, hvor meget ciffertekst der skal til for at bryde et kryptosystem, under den forudsætning at kryptoanalytikeren har uendelig mange ressourcer i form af tid, personale og regnekraft til sin rådighed. Det er som regel sådan, at jo mere ciffertekst man har, des færre af de mulige nøgler vil give en meningsfyldt klartekst. Når man har bestemt, hvor meget ciffertekst der skal til, for at der kun er en mulig fortolkning af cifferteksten, har man bestemt systemets teoretiske sikkerhed. Dette er kryptosystemets **entydighedslængde**[4].

Man kan således beregne entydighedslængden for DES-kryptosystemet som 17.5 tegn, og for Cæsars ciffer til 1,5 tegn [16]. Det betyder, at det teoretisk set er muligt at bryde systemerne hvis, man har henholdsvis 17.5 og 1,5 tegn ciffertekst til rådighed.

Den teoretiske sikkerhed bruges i dag ikke i praksis. Det er der flere grunde til. For det første er det sjældent, at man har uendelige ressourcer til at bryde en kode, og for det andet vil der med de enorme datamængder, der sendes rundt i dag, som regel altid være tilstrækkeligt med ciffertekst til, at koden teoretisk set kan brydes. For det tredje siger entydighedslængden *ikke* altid særlig meget om, hvor svært det er at bryde et system, når man ikke har uendeligt store ressourcer, hvad ingen har i praksis.

[4]fra engelsk unicity distance

### 6.5.2 Beregnelig sikkerhed

I den beregnelige sikkerhed forudsætter man, at der er tilstrækkeligt med ciffertekst, og spørger så : *hvor lang (eller hvor kort) tid vil det tage at bryde koden ?*

Mange systemer i dag – specielt offentlig-nøgle systemer – bygger deres sikkerhed på et kendt matematisk problem, således at det at bryde koden, bliver det samme som at give en algoritme til at løse dette matematiske problem tilstrækkeligt hurtigt. For sådanne systemer kan den beregnelige sikkerhed nemt findes, idet den er givet ved den tid det tager at løse det matematiske problem. Dette kan beregnes ved brug af store $\mathcal{O}$ notationen, som vi beskrev i kapitel 3. Det skal her understreges, at man skal benytte denne med en vis omtanke. Det er der flere grunde til :

1. Kompleksitetsteori giver ofte pessimistiske mål for kompleksiteten. Dette skyldes at man i denne sammenhæng altid beregner kompleksiteten ud fra det værst tænkelige tilfælde.

2. Derudover beskæftiger man sig altid med enkeltstående tilfælde. Et problem kan imidlertid være lette at løse, hvis man arbejder på flere tilfælde på en gang, f.eks hvis man har flere cifferteksters til sin rådighed.

### 6.5.3 Et sikkert system

De fleste systemer kan brydes. Om ikke andet kan man som regel afprøve alle de mulige nøgler i systemet indtil klarteksten kommer frem. Der findes imidlertid et system, der er fuldstændigt sikkert, hvilket vil sige, at uanset mængden af ciffertekst og beregningstid kan koden ikke brydes. Systemet kaldes **engangs-kode**[5] og bygger på bogstavsubstitution. Ideen er, at nøglen er ligeså lang som den meddelelse, der skal indkodes, og at den kun bruges en gang. På denne måde sikres det, at de forskellige tekstblokke aldrig indkodes med den samme nøgle. Systemet fungerer efter følgende procedure :

[5] Fra engelsk **onetime pad**

> Nøglen genereres som en række af tilfældige tal i intervallet 1 til 30 – det danske alfabet plus mellemrum. I denne række af tal skal der være ligeså mange tal, som der er bogstaver i klarteksten, således at hvert tal i rækken kun bruges en gang. Man indkoder så bogstav nummer $n$ i klarteksten ved at finde tal nummer $n$ i rækken og flytte bogstavet et antal pladser til højre alt efter tallet. Med talrækken $28, 5, 22, 14, 21, 1, 19, 7, 6, 17, 3$ bliver klarteksten `MISSISSIPPI` indkodet til `KNKC THPVCL`. Bemærk at her indkodes `S`, som fire forskellige bogstaver.

Grunden til at systemet er totalt sikkert er, at ciffertekstен vil kunne være skabt ud fra en hvilken som helst meningsfyldt klartekst med et passende valg af kode. Selv en afprøvning af alle mulige koder hjælper ikke, idet man så blot vil få genereret alle mulige klartekster af en vis længde. F.eks. vil den fremkomne tekst `KNKC THPVCL` kunne være ciffertekstен svarende til en vilkårlig 11 tegn lang streng i denne rapport, indkodet med en passende nøgle. En anden måde at sige dette på er, at systemet har en uendelig lang entydighedslængde. Dette skyldes, at der til en given ciffertekst altid vil være mange meningsfyldte klartekster.

Årsagen til at systemet ikke ukritisk bruges overalt, er at nøglen bliver meget lang. Systemer som dette, der uanset ressourcer og tid ikke kan brydes, siges at være **totalt sikre**, endnu et begreb defineret af Shannon.

Strømkryptering kan siges at være et forsøg på at efterligne engangskoden, da man ud fra en nøgle, genererer en tilfældig række tal, der tilsammen udgør selve indkodningsnøglen.

## 6.6 Sikkerhed af det omgivende system

Udover at selve den kryptografiske algoritme skal være sikker, skal de ydre rammer som omgiver algoritmen, være mindst ligeså sikre. Det nytter ikke, at man benytter et ubrydeligt kryptosystem, hvis man lader nøglen ligge frit fremme på skrivebordet. Der skal derfor i et kryptografisk system være en fornuftig protokol for administration og distribution af nøgler.

Faktisk kan man sige at kryptografi ikke *fjerner* nogen sikkerhedsproblemer, det *flytter* dem kun til kontrollerbare områder. Fra at skulle

beskytte mange lange tekster, skal man nu kun beskytte få korte nøgler. Sikkerheden af et kryptografisk system er derfor både afhængigt af algoritmens sikkerhed, samt af sikkerheden i nøgleadministrationen.

Nøgleadministration handler om alt fra generering og brug, til sletning af nøgler. Vi har delt det op i de følgende punkter.

1. Generering.
2. Opbevaring
3. Transport
4. Anvendelse.
5. Sletning.

En særlig type problem udgøres af de offentlige nøgler.

### 6.6.1 Sikkerhed i nøglegenerering

For at skabe sikre kryptosystemer er det naturligvis vigtigt at skabe sikre nøgler. En sikker nøgle er dels en "tilfældig" nøgle, og dels en nøgle der ikke er svag, i forhold til det kryptografiske system.

Det at en nøgle skal være tilfældig, betyder at udefra kommende personer ikke må kunne gætte den eller beregne sig frem til den, ud fra kendskab til systemet, eller de personer der kender nøglen. Hvis brugerne af kryptosystemet selv vælger deres nøgler, vil de som regel vælge bogstavkombinationer, de let kan huske – fødselsdage, børnenes navne eller lignende. Vælger man nøglerne således, indskrænker man antallet af sandsynlige nøgler væsentligt, hvilket gør det lettere at bryde krypto-systemet. Hvis det er kryptosystemet der vælger nøglen, skal dette gøres tilfældigt, således at en udenforstående person ikke kan genskabe nøglegenereringsprocessen. Man bør f.eks. ikke bruge datoen eller antallet af sekunder siden midnat som inddata til en tilfældigheds generator, da det giver for få mulige nøgler, og for datoens vedkommende også for let gennemskuelige nøgler.

En anden vigtig ting er at nøglen ikke er 'svag' i forhold til kryptosystemet. I Cæsars ciffer er nøglerne 0, 30, ... svage, da de gør cifferteksten og klarteksten ens. De fleste kryptosystemer vil have den slags svage nøgler, ud fra hvilke klarteksten let kan genskabes.

En sidste vigtig ting ved nøglegenerering er at sikre det miljø det foregår i. Det nytter jo ikke noget at lave sikre nøgler, hvis det foregår i fuld offentlighed.

### 6.6.2 Sikkerhed i opbevaring

Den enkleste måde at gemme en nøgle på – set fra en nøgleadministrators side – er ved at lade brugeren opbevare den.

En anden metode til at opbevare nøglerne på, er ved at gemme den i speciel sikker hardware. Det er hardware som en udenforstående ikke kan få adgang til, og som det ikke er muligt at aflytte eller på anden måde få oplysninger ud af. Ofte skal et kryptosystem imidlertid køre på en almindelig flerbrugermaskine, hvor sådant hardware ikke findes. Man kan så støtte sig til den almindelige software-baserede beskyttelse, der findes i sådanne systemer, men skal i så fald være meget opmærksom på evt. faldgruber f.eks. i form af superbrugere, der har ekstraordinære privilegier.

Den tredje mulighed er at beskytte kryptografien ved hjælp af kryptografi. Nøglerne er data som alle andre meddelelser, og kan derfor kryptograferes. Det kan ske ved at alle nøglerne kryptograferes med en **nøgleindkodningsnøgle**. Dette kan eventuelt foregå på flere niveauer, sådan at man til sidst kun skal beskytte en enkelt nøgle. Denne ene nøgle skal dog også beskyttes, og det kan kun ske med en af de ovenstående metoder. Nøgleindkodningsnøgler på de lavere niveauer kaldes også for **sessionkeys**, mens hoved-nøgleindkodningsnøglen kaldes for en **hovednøgle** eller en **masterkey**.

### 6.6.3 Transport

Problemerne ved transport af nøgler, er stort set de samme, som ved opbevaring.

En mulighed for transport af nøglerne er at sende at sende dem i kryptograferet tilstand. Igen opstår der imidlertid det problem, at til sidst kan denne metode ikke længere anvendes, da hovednøglen må transporteres på anden vis. Det er dog et meget mindre problem, fordi det kun drejer sig om en nøgle. Her kan hybridsystemer anvendes.

En anden mulighed til at distribuere nøglerne er ved at benytte en af de gode gamle metoder – at sende nøglerne med en betroet kurer[6]. Kureren kan imidlertid overmandes eller på anden måde frarøves nøglerne.

Bedre er det som ovenfor at bruge speciel sikret hardware. Det kan f.eks. være i form af en transportabel **nøgle-kanon**, der kan modtage eller generere, opbevare og afgive nøgler. Kanonen bør være designet således, at den ikke giver nøglerne til uautoriserede personer/maskiner, men kun til dem de er tiltænkt. Maskinen bør derfor kunne identificere sådanne. Udover at kanonen skal være designet således, at der ikke kan brydes ind i den, skal den også være designet, så dens funktion og udseende ikke kan efterlignes af udenforstående.

### 6.6.4 Sikkerhed under brug af nøgler

Ved brugen af nøgler gælder mange af de samme ting som ved transport og opbevaring, men med en vigtig undtagelse : nøglerne *skal* være i klartekst, når de anvendes. Dette problem kan enten løses i hardware eller i software.

Hardwareløsningen består som tidligere nævnt af et specielt sikkert stykke hardware. Heri opbevares hovednøglen, hvis der er nogen, og heri foregår nøglegenerering, samt ind- og afkodning. Nøglerne bør ikke findes i klartekst udenfor dette stykke hardware, så enten ankommer de krypteret, eller også opbevares de her konstant.

I softwareløsningen må man som før nævnt tage udgangspunkt i de muligheder, operativsystemet tilbyder for beskyttelse af data. Dette varierer fra system til system, og der kan derfor ikke siges så meget generelt, udover at krypteringsprogrammet bør køre med så megen beskyttelse som overhovedet muligt. Se [16, side 191-330].

### 6.6.5 Sikkerhed under og efter sletning af nøgler

Alle kryptograferede nøgler bør skiftes med mellemrum. Enten fordi de er gået tabt, dekrypteret eller lignende, eller for at gøre livet surt for kryptoanalytikerne. I de situationer hvor den gamle nøgle "slettes" bør man sikre sig at den også vitterligt slettes. F.eks. sletter mange operativsystemer filer blot ved at markere pladsen som slettet, uden at de gamle data overskrives. Det kan gøre det muligt for udenforstående

[6] Dog ikke nødvendigvis til hest.

at læse indholdet af en slettet fil. Derfor bør filer, hvor en nøgle har været opbevaret ikke blot slettes, men pladsen skal også overskrives med tilfældige data. [8] anbefaler at det foretages ca. 1000 gange, for at man kan være helt sikker på at den ikke kan læses. Derudover bør man også sikre sig at nedskrevne kopier er destrueret f.eks. makuleret.

### 6.6.6 Sikkerhed af offentlige nøgler

De offentlige nøgler i et offentlig-nøgle system skal naturligvis ikke holdes hemmelige. Der er derfor ikke de samme problemer med at holde dem hemmelige under opbevaring, transport og brug. Der er derimod et andet problem. Man skal sikre sig at nøglen vitterligt kommer fra den person der påstår at have sendt den. F.eks. kan en kryptoanalytiker udgive sig for en anden person A, og distribuere en nøgle som hun påstår kommer fra A. Hvis andre tror på det, og begynder at indkode med denne nøgle, har kryptoanalytikeren ingen problemer med at genskabe klarteksterne.

En metode til at løse dette problem er ved at overlade nøgleudvekslingen til brugerne. Så er de selv ansvarlige for at de nøgler som anvendes er korrekte. En anden mulighed er at have en eller flere **nøgleadministratorer**, folk som administrerer de offentlige nøgler. En nøgleadministrator er ansvarlig for, at alle nøgler er korrekte. Han kan eventuelt også være ansvarlig for at generere dem. Hvis en person skal bruge en offentlig nøgle, får han den udleveret af nøgleadministratoren.

### 6.6.7 Organisatoriske spørgsmål

Selvom der findes mange midler til at gøre kryptografiske systemer sikre, afhænger sikkerheden i sidste ende af, at man kan stole på de mennesker, som designer, implementerer og bestyrer kryptosystemet.

Det er derfor, som vi også skrev i kapitel 5 om datasikkerhed, vigtigt at organisationen omkring kryptosystemet kan leve op til opgaven. Sløseri med nøgler og almindelig uhæderlighed har store konsekvenser, netop fordi sikkerheden alene afhænger af disse.

## 6.7 Anvendelse af kryptografi i edb-systemer

Kryptografi som middel til sikring af fortrolighed er relevant overalt, hvor der findes data i et edb-system. Vi vil i det følgende dele edb-systemer i 4 grupper, hver med deres særlige krav til et givet kryptosystem.

1. Indlæsning af data samt transmission mellem terminal og datamaskine.
2. Databehandling i datamaskinen.
3. Lagring på baggrundslager samt transmission mellem datamaskine og baggrundslager.
4. Kommunikation mellem datamaskiner via netværk.

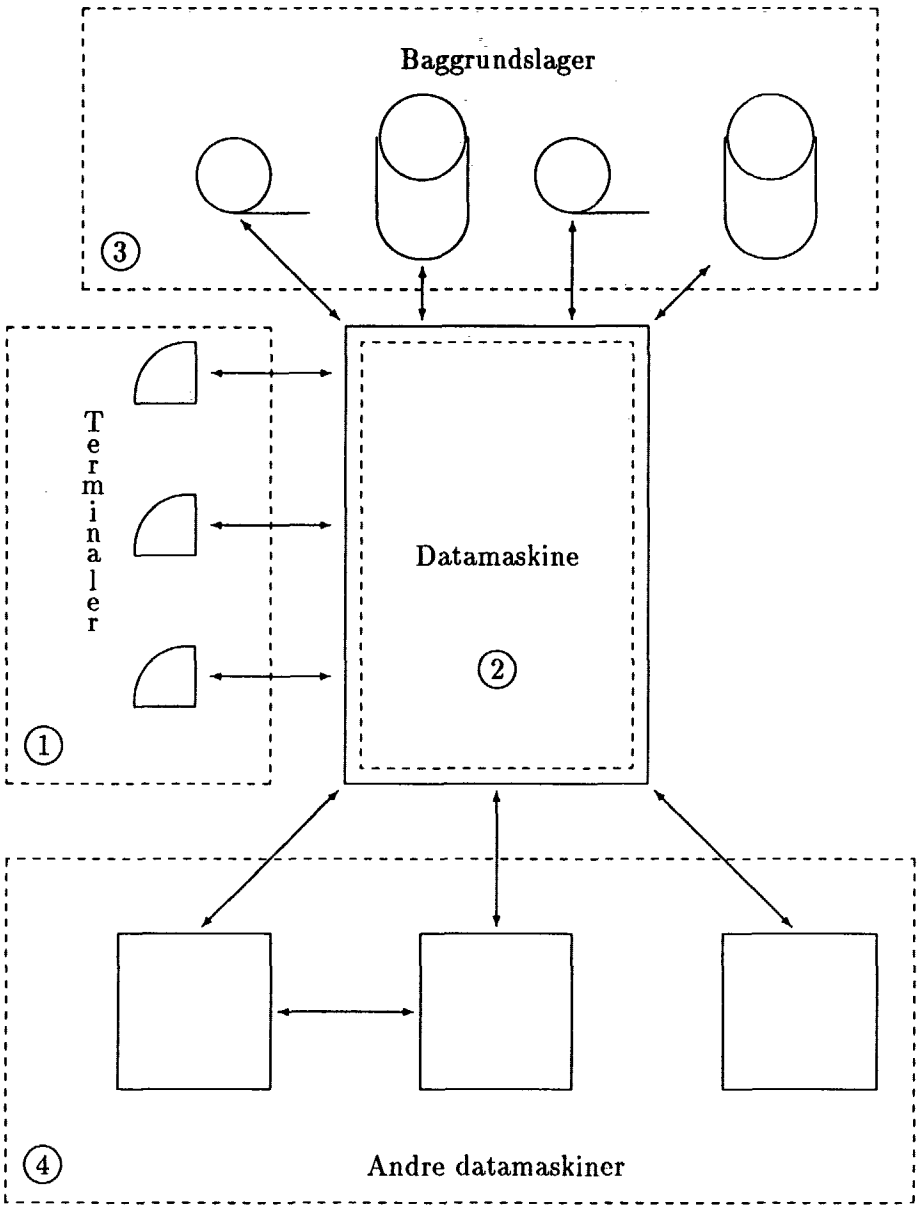
Denne opdeling er anskueliggjort på figur 6.7.

### 6.7.1 Indlæsning af data

Når data første gang kommer ind i et edb-system, sker det som regel via terminaler, måleapparater eller andet lignende udstyr. Hvis denne del af udstyret samt transmissionslinien til centralenheden ikke er beskyttet, giver det udefra kommende personer mulighed for at tappe oplysninger. For at undgå dette kan man blandt andet beskytte udstyret ved brug af kryptografi.

Denne anvendelse af kryptografi har den ulempe, at terminaler oftest er ret 'dumme', forstået på den måde, at de ikke indeholder nogen mulighed for generel databehandling. Skal der gives mulighed for kryptografi, må der derfor installeres ekstra hardware. Der er ikke specielt hårde krav til ind- og afkodningshastigheden, da terminaler og centralenheder kommunikerer forholdsvis langsomt, oftest 9600 bits/sekund.

Hvis der kun er behov for at beskytte inddata fra terminalerne, kan sikkerhedsspørgsmålet løses elegant med offentlig-nøgle kryptering. Forbindelsen centralenhed – terminaler er en en-til-mange forbindelse (en centralenhed til mange terminaler). Derfor vil der kun være behov for et sæt nøgler med et offentlig-nøgle system. Den hemmelige nøgle




**Figur 6.7:** Kryptografi i et edb-system

lagres på centralenheden, hvorefter alle terminalerne kan udstyres med den samme offentlige nøgle. Derved forsvinder behovet for fortrolig kommunikation helt. Samtidig skal de 'dumme' terminaler kun kunne indkode, idet nøglegenerering og afkodning altid foregår centralt. Hvis der yderligere stilles krav til, at uddata skal krypteres, kan denne metode ikke benyttes. Offentlig-nøgle kryptering er dog stadig relevant, da det fjerner behovet for udveksling af hemmelige nøgler.

### 6.7.2 Databehandling

Efter at data er indtastet og ankommet til centralenheden, vil de befinde sig i maskinens hukommelse, og blive manipuleret af et program. Her er behovet for beskyttelse naturligvis ligeså stort, som udenfor centralenheden. Dette skyldes, at programmer kan overvåges, og indholdet af maskinens registre aflæses.

Man kan blandt andet beskytte data ved at pakke maskinen ind i en **krypteringsboks** – en kasse som ikke kan åbnes uden at alle data slettes. Dette er dog ikke altid muligt, f.eks hvis de pågældende data skal anvendes i et almindeligt flerbrugersystem. En anden mulighed er at have tillid til maskinens indbyggede sikkerhedssystemer i form af rettighedslister med mere. De senere års erfaringer med hackere viser dog, at dette ikke altid er tilstrækkeligt. Derfor kan det være nødvendigt at beskytte data på en anden måde. F.eks. ved brug af kryptografi.

Det særlige ved denne anvendelse af kryptografi er at der ikke finder nogen transmission sted, som ved de andre anvendelser. Derfor er der ikke nogle problemer med nøgledistributionen.

Et andet særkende er, at der ved kørslen af et program skal udføres operationer på de involverede data. Dette *kan* være en faktor, der umuliggør anvendelsen af kryptografi. Det er nemlig langt fra alle kryptosystemer som bibeholder data'enes egenskaber med hensyn til addition, multiplikation, ordning osv. Beregninger foretaget på krypterede data, skulle jo gerne give det samme resultat som beregningerne foretaget på de ikke-krypterede.

Mere præcist skal der gælde at :

*Databehandling (Indkodning (klartekst)) =*
*Indkodning (Databehandling (klartekst))*

Med de rette kryptografiske algoritmer, kan dette godt opfyldes for de almindelige regningsarter. Ønsker man derimod at foretage sorteringer og sammenligninger, får man imidlertid et problem. Arbejder man f.eks. med de hele positive tal som klartekster, og ønsker at kunne benytte sammenligningsoperatoren '<', kan kryptoanalytikeren ofte bryde systemet. Se [16, side 157-161].

### 6.7.3 Lagring af data

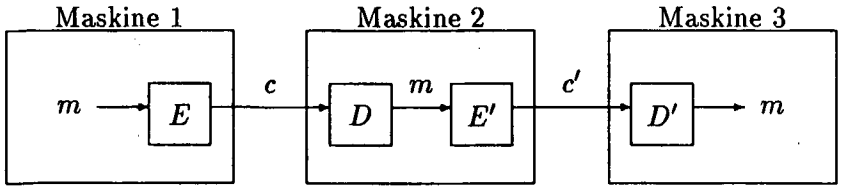
Efter indlæsning af data og afsluttet databehandling, vil der som regel være behov for at lagre data. Her er der naturligvis også mulighed for at få fat på fortrolige oplysninger, enten under transmissionen mellem centralenhed og lagringsmedie, eller mens data ligger lagret.

Denne anvendelse af kryptografi kræver hurtige algoritmer, hvis ikke lagring og læsning skal forsinkes voldsomt. Oftest transmitteres data mellem centralenhed og pladelagre nemlig med millioner af bits pr sekund. Derudover skal der være mulighed for at ind- og afkode små stumper tekst af gangen. Dette er f.eks nødvendigt ved anvendelser af databaser, da der er behov for at kunne hente og lagre gemme enkelte poster, uden at *hele* databasen skal ind- og afkodes hver gang. En sidste ting der er værd at bemærke, er at de krypterede data transmitteres fra centralenheden via lagringsmediet og tilbage til den samme centralenhed til afkodning. Da ind- og afkodning således foregår samme sted, er der ingen problemer med nøgledistribution.

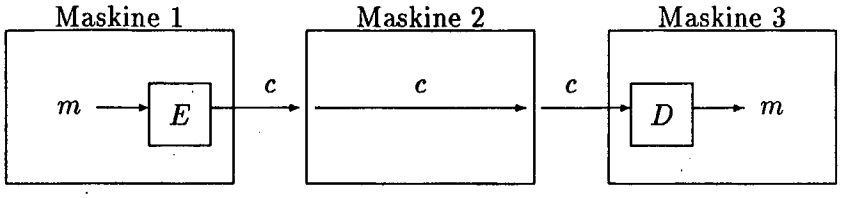
### 6.7.4 Datakommunikation

Det sidste anvendelsesområde for kryptografi er i datakommunikation i et netværk, f.eks. i forbindelse med elektronisk post. Forskellen på dette anvendelsesområde og det først beskrevne – transmission fra terminal til centralenhed – er, at der her er tale om kommunikation mellem edb-maskiner, og ikke mellem en edb-maskine og 'dumme' terminaler. Desuden er der tale om kommunikation mellem mange maskiner på kryds og tværs, og ikke kun mellem en maskine og mange perifere enheder. Se figur 6.7.

I et større netværk vil det være uoverkommeligt at forbinde alle maskiner direkte med hinanden. Ofte vil data passere en hel række maskiner,




**Figur 6.8:** Link kryptering




**Figur 6.9:** End-to-end kryptering

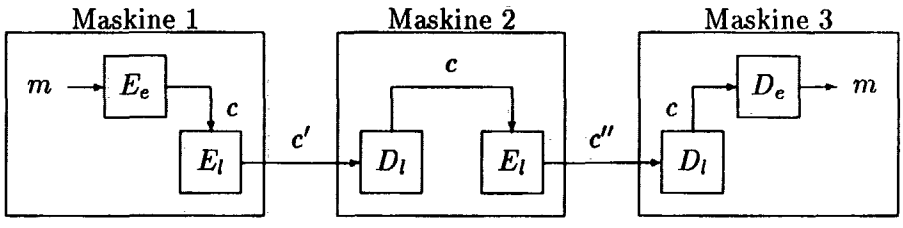
før de når frem til bestemmelsesstedet. Man skelner derfor mellem to typer kryptering i et netværk :

- Link-kryptering.
- End-to-end kryptering.

### Link-kryptering

I link-kryptering ind- og afkodes meddelelserne ved hver maskine mellem start og slutpunkterne, som vist på figur 6.8.

Det har den fordel, at kryptering kan foregå fuldstændig transparent for brugeren. Ofte vil ind- og afkodningsfaciliteten være en boks, der er koblet direkte på transmissionslinien, der hvor den er forbundet med datamaskinen. En anden fordel er, at nøgleadministrationen bliver relativt enkel. Der er hele tiden tale om kun to maskiner, der kommunikerer med hinanden. Ulempen er, at meddelelserne eksisterer i klartekst i de enkelte maskiner undervejs.




**Figur 6.10:** Kombineret end-to-end og link-kryptering

### End-to-end kryptering

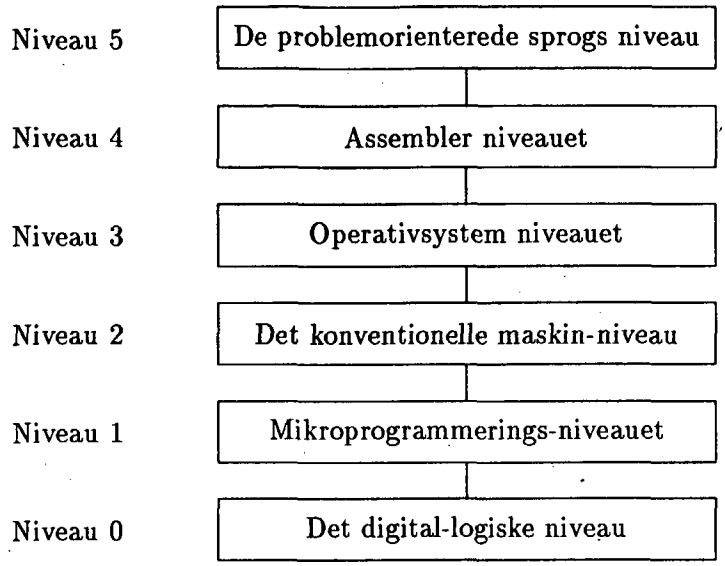
Her indkodes meddelelsen ved startmaskinen og holdes indkodet indtil slutmaskinen nås, som vist på figur 6.9. Dette bevirker, at systemet er mere sikkert end ved link- kryptering. Til gengæld får man et problem med nøgleadministrationen, da mange maskiner skal kommunikere fortroligt to og to. Dette kræver som tidligere nævnt at de to har en fælles nøgle, hvilket i et hemmelig-nøgle system betyder at nøgleantallet stiger kvadratisk med antallet af brugere i systemet.

En anden ulempe ved end-to-end kryptering er at den muliggør analyse af kommunikationsstrømmene. Denne analyse kan afsløre til hvilken maskine de krypterede meddelelser sendes. I link-kryptering kan hele meddelelsen inklusive adressen holdes hemmelig mellem maskinerne, da den afkodes ved hver enkelt maskine på vejen, hvorefter adressen kan aflæses af maskinen og indkodes igen. I end-to-end kryptering, bliver man nødt til at opbevare slut-adressen i klartekst, da der ikke foregår afkodning ved hver maskine. En mulig løsning på problemet er at kombinere de to typer kryptering, som vist på figur 6.10.

I end-to-end kryptering kan systemet både laves brugerstyret og transparent for brugeren, som ved link-kryptering. Da data holdes indkodet under hele transmissionen, foretrækkes end-to-end kryptering til elektronisk post og pengeoverførsler.

## 6.7.5 Niveauer

Som afslutning på dette afsnit vil vi se på, hvordan kryptografi kan placeres i henholdsvis den enkelte datamaskine og i et netværk.




**Figur 6.11:** Model af en datamaskine

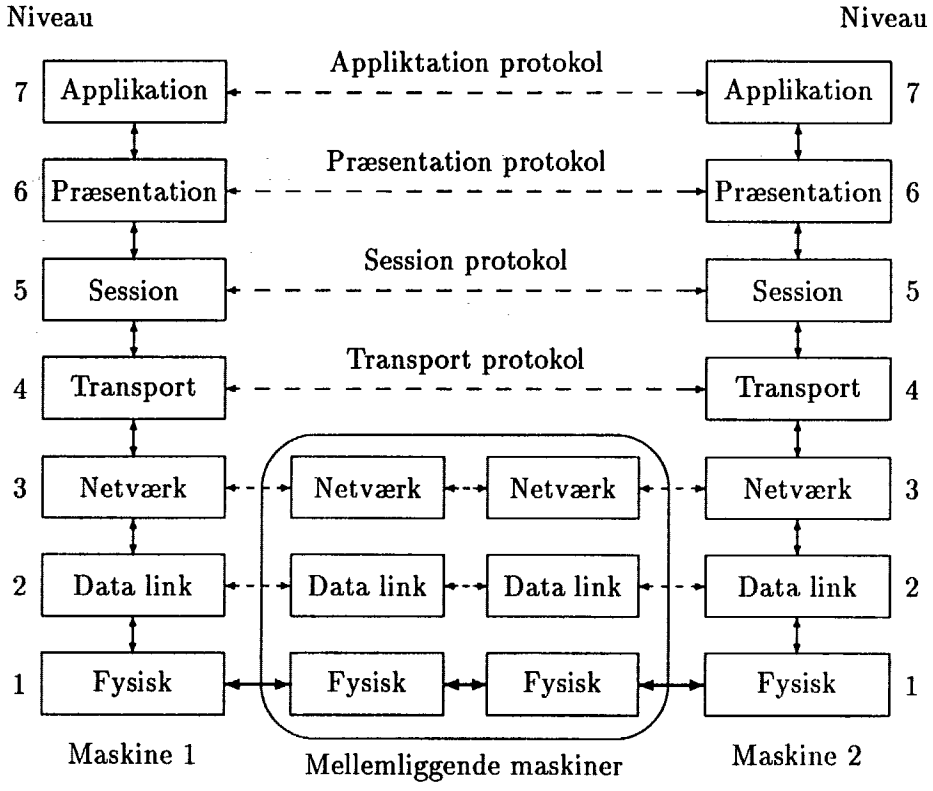
En datamaskine kan opdeles som vist på Tannenbaums [54] 6-trins model, figur 6.11. Her ligger de kryptografiske faciliteter enten sammen med de øvrige sikkerhedssystem *i* selve operativsystemet eller, hvad der er det mest almindeligt ovenpå, som en del af de forskellige applikationsprogrammer.

Et netværk kan på tilsvarende vis struktureres i en 7-lags model, som vist på figur 6.12. Det er den såkaldte OSI[7]-model. Se [8] og [55].

For begge modeller gælder, at de 6 / 7 lag er overbygninger på det underliggende lag, og for OSI-modellen gælder at hvert lag kommunikerer med det tilsvarende lag på andre maskiner. Hvert lag er dedikeret til at udføre specielle opgaver i forbindelse med kommunikationen, således at der på de laveste lag, udføres de mest hardware-specifikke opgaver, mens der på de højere lag udføres operativsystem- og netværksspecifikke opgaver uafhængigt af den underliggende hardware.

For kryptografi gælder, at hvis det foregår i et lag højt oppe i modellen, vil det være uafhængigt af netværkets fysiske karakteristika, såsom

[7]Open Software Interconnection




**Figur 6.12:** Model af et netværk

liniens udseende, og hvilke maskiner der fysisk er koblet sammen. Eksempelvis foregår end-to-end kryptering og specielt brugerstyret kryptering her. Centralenhed-til-terminal kryptering foregår længere nede i hierarkiet, men kan stadig godt være uafhængig af transmissionslinieens karakteristika. Link-kryptering foregår på de laveste niveauer. Her krypteres der mellem fysiske forbindelser i netværket, og meddelelserne er kun synlige op til niveau 3. "Ægte" link-kryptering, hvor det er selve liniekommunikationen der indkodes, ligger helt nede på det digital logiske niveau.

## 6.8 Opsamling

Dette kapitel har beskrevet hvad kryptologi er, og vist hvilke problemer der kan opstå når det skal anvendes i praksis.

Vi har vist at offentlig-nøgle kryptografi, som jo et hoved-emne i denne rapport, primært har to anvendelser. Dels som et selvstændigt kryptosystem til netværkskommunikation, hvor det kan nedbringe antallet af nøgler til en brøkdel af hvad et hemmelig-nøgle system ville kræve, og dels som den ene del af et hybrid-system, hvor det kan fjerne de nøgleudvekslingsproblemer der ellers er i hemmelig-nøgle systemer. I næste kapitel beskriver vi andre anvendelser af offentlig-nøgle systemer.

# Kapitel 7

# Anvendelser af offentlignøgle kryptografi

## 7.1 Envejs- og smæklåsfunktioner

Envejs- og smæklåsfunktioner er vigtige begreber i forbindelse med offentlig-nøgle kryptografi.

### 7.1.1 Envejsfunktioner

En envejsfunktion kan defineres som følger [5] :

**Definition 7.1 (Envejsfunktion)**
*Givet to vilkårlige mængder $X$ og $Y$, er en funktion $\mathcal{E} : X \rightarrow Y$ envejs, hvis der gælder*

1. *$y = \mathcal{E}(x)$ er en letberegnelig for alle $x \in X$.*
2. *$x = \mathcal{E}^{-1}(y)$ er umulig eller meget vanskelig at beregne for næsten alle $y \in Y$.*

En envejsfunktion er ikke det samme som en ikke-invertibel funktion. Det vigtige i definitionen af en envejsfunktion er, at det er *beregningsmæssigt* svært eller umuligt at finde $x = \mathcal{E}^{-1}(y)$.

Et eksempel på en envejsfunktion er

$$y = \mathcal{E} = a^x \text{ MOD } b$$

Selv når man kender $y, a$ og $b$, er det beregningsmæssigt meget svært at finde et $x$ der opfylder ligningen [5].

### 7.1.2 Smæklås-funktioner

Til offentlig-nøgle kryptografi har man brug for en funktion, der har nogle af de samme egenskaber som en envejsfunktion. Det kræves at funktionen er let at beregne samtidig med, at den inverse kun må kunne beregnes effektivt, givet en nøgle. En sådan funktion kaldes en smæklås-funktion :

**Definition 7.2 (Smæklåsfunktion)**
*En funktion $S : X \rightarrow Y$ siges at være en smæklås-funktion, hvis den er en envejs funktion, der let kan inverteres givet en særlig ekstra information.*

At sådanne funktioner kaldes smæklås-funktioner, kommer naturligt nok af at en smæklås er en lås, som alle kan låse, men som kun kan åbnes af dem der har en nøgle. Den primære anvendelse af smæklås-funktioner findes i offentlig- nøgle kryptografi, som gennemgås i detaljer i de næste to kapitler.

Et simpelt eksempel på en smæklåsfunktion er multiplikation af primtal. Det er meget let at multiplicere selv meget store tal med hinanden, men det er meget svært at finde de faktorer som et stort tal er sammensat af. Enhver kan for eksempel multiplicere 127 og 243, og få 30861, men hvilke to primtal går op i 31313 ? Kender man imidlertid også $\phi(n)$, kan primtallene let beregnes. Se afsnit 9.1.3.

## 7.2 Anvendelser af envejs-funktioner

### 7.2.1 Etablering af hemmelig nøgle over offentlig linie

Den største svaghed ved kryptosystemer baseret på en fælles hemmelig nøgle er, at nøglen på en fortrolig måde skal kunne udveksles inden de involverede parter kan påbegynde transmission af de kryptograferede meddelelser. Denne prækrypto-transmission er yderst sårbar og *kan* være en begrænsende faktor for kryptosystemets anvendelighed.

Anvendelsen af envejsfunktioner muliggør, at to parter A og B kan opnå kendskab til en fælles hemmelig nøgle, uden at have anden kontakt end en offentlig kommunikations kanal. Dette kan gøres uden, at nøglen kan opsnappes af en tredie part.

Systemet blev foreslået af Diffie og Hellman i 1976 [17], og er det første forslag til noget, der ligner et offentlig-nøgle system. Det er dog ikke et rigtigt kryptosystem, da det ikke kan ind- og afkode en meddelelse. Derimod udgør systemet en protokol til etablering af en *fælles* hemmelig nøgle over en offentlig linie. Det kaldes derfor i stedet et offentligt-nøgle-distributions system og bygger på en enkel protokol, der skal udføres af de to implicerede parter A og B.

1. A og B bliver (i fuld offentlighed) enige om to heltal $a$ og $n$ samt envejs-funktionen $\mathcal{E}_{a,n}(x) = a^x \bmod n$. Bemærk at $a$ og $n$ er faste tal.

2. (a) A vælger et tilfældigt heltal $x$ og beregner $c_x = a^x \bmod n$.

   (b) B vælger et tilfældigt heltal $y$ og beregner $c_y = a^y \bmod n$.

3. A og B udveksler $c_x$ og $c_y$ over den offentlige kanal, *men holder hhv $x$ og $y$ hemmelige* (kun A kender $x$ og kun B kender $y$).

4. A beregner så

$$c_{xy} \equiv c_y^x \equiv a^{yx} \bmod n$$

   B beregner så

$$c_{xy} \equiv c_x^y \equiv a^{xy} \bmod n$$

Herefter kender både A og B værdien af $c_{xy}$, selvom den aldrig har været transmitteret over den offentlige kanal. Da de nu har etableret den hemmelige nøgle, kan $x$ og $y$, der skulle holdes hemmelige, smides væk.

En tredie part, der opsnapper hele transmissionen, kan ikke finde $C_{xy}$ med mindre $\mathcal{E}_{a,n}$ kan inverteres, hvorved $x$ og $y$ kan beregnes.

### 7.2.2 Password-beskyttelse

Et password kan betragtes som en brugers nøgle til et edb-system. Hver gang han skal koble sig ind på systemet, skal han for eksempel opgive sit (offentlige) brugernummer og sit (hemmelige) password. EDB-systemet slår derefter op i en tabel over brugernumre og passwords, og ser om det opgivne brugernummer og password passer sammen.

Hvis passwords og brugernumre bliver lagret uden beskyttelse, kan en udefra kommende person få adgang til tabellen over brugernumre og passwords. Dette betyder, at han får adgang til hele edb-systemet, incl. alle data lagret på det. Derfor er det nødvendigt, at tabellen med passwords og brugernumre er ekstra godt beskyttet mod uautoriserede indtrængere.

En metode til at beskytte passwords på, er ved at kryptografere dem. Men det kræver, at en hemmelig nøgle konstant skulle ligge lagret på systemet. En afsløring af den hemmelige nøgle, ville føre til dekryptering af hele tabellen, og kompromittere *alle* brugere og alle oplysninger på systemet. En bedre metode er derfor at benytte envejs-funktioner[5]. Tænk på at edb-maskinen i virkeligheden ikke har behov for at kende de forskellige password : den skal blot kunne checke at de er korrekte. Disse to øjensynligt modstridende ønsker, kan opfyldes med en vilkårlig envejs-funktion $\mathcal{E}$.

- Når en bruger ønsker at sætte eller ændre sit password, indkodes det nye password med $\mathcal{E}$, og det indkodede password gemmes i en tabel sammen med brugernummeret. Det originale password huskes ikke af maskinen.

- Når brugeren vil koble sig på systemet, opgiver han sit brugernummer og password. Passwordet indkodes som før med $\mathcal{E}$, og checkes sammen med brugernummeret i tabellen mod den lagrede tabelværdi.

Bemærk at et glemt password ikke kan genskabes, hverken af maskinen, der ikke kender det, eller af brugeren, der har glemt det. Af samme grund kan en udefra kommende person heller ikke genskabe det ud fra tabellen.

### 7.2.3 Plat og krone over telefonen

Som et sidste eksempel på anvendelse af envejs-funktioner vil vi beskrive, hvordan man kan slå plat og krone over telefonen.

Problemet kan beskrives som følger :

> To personer A og B skal slå plat og krone over telefonen. F.eks. fordi de er blevet skilt og bor i hver sin by. De skal så afgøre, hvem der skal have bilen og hvem der skal have børnene. Hvis A kaster mønten, kan B ikke kontrollere om A taler sandt, når hun siger "*Nu kaster jeg, ... og det blev plat*". Med denne metode kan A vælge det udfald, der passer hende bedst. Derved får hun enten bil eller børn, helt efter hendes eget ønske.

For at forhindre snyderi, er det nødvendigt, at A ikke kan kaste mønten igen, efter at B har gættet, samtidig med at B ikke må kende udfaldet *før* han gætter.

Problemet kan løses med en envejs-funktion [5].

Man skal bruge

1. To mængder $X$ og $Y$ der begge indeholder lige mange lige og ulige tal.

2. En envejsfunktion $\mathcal{E} : X \rightarrow Y$, som A og B skal være enige om.

A og B udfører derefter følgende protokol :

1. A vælger et tilfældigt tal $x \in X$, Beregner $y = \mathcal{E}(x)$ , og sender $y$ til B.

2. B gætter nu om $x$ er lige eller ulige, og fortæller A sit gæt.

3. A kan nu fortælle B, om han gættede rigtigt, ved at se på $x$ som hun opbevarer.

4. B kan checke om A fortæller sandheden, ved at få $x$ fra A, og beregne $y' = \mathcal{E}(x)$, og checke om $y' = y$.

Bemærk at B først kan checke at gættet er rigtigt, efter at han har fået $x$, dvs efter at han har gættet en løsning. Samtidig har A bundet sig til $x$ ved at beregne $y$, og sende den til B.

Metoden kræver at envejs-funktionen $\mathcal{E}$ opfylder visse betingelser. Man må ikke ud fra $y$ kunne se, om $x$ er lige eller ulige. Derudover skal $\mathcal{E}$ være injektiv, således at :

$$x_1 \neq x_2 \Rightarrow \mathcal{E}(x_1) \neq \mathcal{E}(x_2)$$

Er dette ikke tilfældet, kunne man tænke sig, at A fandt et lige $x_1$ og et ulige $x_2$, så $\mathcal{E}(x_1) = \mathcal{E}(x_2)$, således at hun ikke havde lagt sig fast på noget udfald, før B havde gættet.

## 7.3 Anvendelser af smæklås-funktioner

Som nævnt er det indenfor offentlignøgle systemer, at smæklåsfunktionerne bliver anvendt. I dette afsnit ser vi på nogle generelle anvendelser af offentlig nøgle systemer.

Følgende symboler anvendes :

| | |
|---|---|
| Offentlig kodningsfunktion (en smæklås) | $E$ |
| Hemmelig kodningsfunktion (den inverse) | $D$ |
| Kodet meddelelse (Ciffertekst) | $c_h$ |
| Signeret Meddelelse | $c_s$ |
| Kodet *og* signeret meddelelse | $c_{sh}$ |

### 7.3.1 Digital underskrift med offentlignøgle systemer

Hvis man vil sikre autenticiteten af en meddelelse, skal man sørge for, at *modtageren* kan være sikker på, hvem der har sendt meddelelsen. Hvis man ydermere har brug for at overbevise andre om, hvem der har sendt en meddelelse, taler man om **digitale underskrifter** eller **digitale signaturer**, dvs. en elektronisk underskrift der er ligeså bindende som en rigtig underskrift.

Man kan f.eks. forestille sig, at A har sendt B følgende meddelelse "`Jeg skylder dig (B) 1000 kr.`" For at kunne bruge dette i en retssag skal B, når A ikke betaler, kunne overbevise dommeren om, at A og kun A kan have sendt meddelelsen.

En metode til at sikre dette er ved brug af et offentlignøgle system :

- Man skal benytte et offentlig-nøgle system hvor mængden af klartekster er lig mængden af ciffertekster : $M = C$.
- Afsenderen A viser sin identitet overfor modtageren B ved at *indkode* hele meddelelsen (eller blot en speciel del deraf) med *afkodnings*algoritmen $D$ og sin hemmelige nøgle $k_{D_A}$ :

$$c = D(m, k_{D_A})$$

- Når B skal *afkode* meddelelsen, sker det med A's offentlige *indkodnings*nøgle $k_{E_A}$ :

$$m = E(c, k_{E_A}) = E(D(m, k_{D_A}.), k_{E_A})$$

- Når B modtager meddelelsen, ved han, at kun A kan have sendt den, da kun A kender den hemmelige afkodningsnøgle $k_E$.

Bemærk at med denne metode sikres fortroligheden ikke, idet enhver kan afkode den krypterede meddelelse. Dette skyldes at afkodningen kun kræver den offentlige nøgle $k_{E_A}$. Ønsker man udover identifikationen også at sikre fortroligheden i systemet, udbygger man det på denne måde :

- Afsenderen A indkoder først sin meddelelse med sin egen hemmelige nøgle $k_{D_A}$ som før. Derefter indkodes resultatet med B's offentlige nøgle $k_{E_B}$ :

$$c = E(D(m, k_{D_A}), k_{E_B})$$

- Da meddelelsen er indkodet med B's offentlige nøgle, er det nu kun B der kan afkode meddelelsen :

$$\begin{aligned} m &= E(D(c, k_{D_B}), k_{E_A}) \\ &= E(D(E(D(m, k_{D_A}), k_{E_B}), k_{D_B}), k_{E_A}) \\ &= E(D(m, k_{D_A}), k_{E_A}) \\ &= m \end{aligned}$$

Man pakker således signaturen ind i hemmeligholdelsen. Et billede herpå er, at man først underskriver et brev og derefter lægger det i en forseglet konvolut, inden det afsendes.

Når A har indkodet den ovenstående meddelelse $c = D(m, k_{D_A})$, og sendt den til B, har B et tilstrækkelig godt bevis, idet A og kun A kender $k_{D_A}$. Afkodes cifferteksten $c$ til den ovenstående meddelelse med A's offentlige indkodningsnøgle, er sagen bevist.

Metoden virker også, selvom man kun signerer en lille del af en meddelelse, f.eks. de sidste par linier. Afsenderen og modtageren skal da være enige om hvilken del af meddelelsen, der er signeret, så det kun er den del, der bliver "dobbelt-afkodet".

### 7.3.2 Poker

"Mental-poker" (fra [16, side 100-115]) er ligesom den digitale underskrift et eksempel på, at offentlignøgle-systemer kan bruges til andet end hemmeligholdelse. Det spilles som almindelig poker, blot med den forskel at kortene er erstattet af meddelelser. Det betyder at de regler, der gælder for almindelige kortspil, skal overholdes :

- Hver spiller må kun kende sine egne kort – sin egen "hånd".
- Alle "hænder" er lige sandsynlige.
- Spillernes hænder skal være indbyrdes disjunkte. Et kort kan kun være hos en spiller ad gangen.
- Når spillet er slut, skal det være muligt at kontrollere at ingen har snydt.

Spillet går som almindelig poker ud på, at hver spiller får 5 kort til start. I løbet af spillet kan kortene byttes ud med kort fra bunken af ubrugte kort. Vinder er den spiller, der til sidst sidder med den bedste hånd, defineret efter regler vi ikke vil komme ind på her.

Idet vi for nemheds skyld begrænser os til 2 personer A og B, spilles mental poker således :

Først opretter hver person et sæt nøgler $(e_A, d_A)$ og $(e_B, d_B)$, hvor der specielt skal gælde at

$$E_A(E_B(m, e_B), e_A) = E_B(E_A(m, e_A), e_B)$$

Til hvert af de 52 kort knyttes så en meddelelse :

$$\begin{aligned} m_1 &: \text{“Kløver 2”} \\ m_2 &: \text{“Kløver 3”} \\ &\vdots \\ m_{51} &: \text{“Spar konge”} \\ m_{52} &: \text{“Spar es”} \end{aligned}$$

Spillet kan nu gå igang.

B blander kortene ved at indkode $m_1, \ldots, m_{52}$ :

$$c_i = E(m_i, e_B) \quad i = 1, \ldots, 52$$

og derefter bytte $c_i$'erne rundt. Han sender nu det indkodede og blandede spil kort til A.

A “giver” ved at vælge 5 tilfældige “kort” $(c_{B1}, \ldots, c_{B5})$ og sende dem til B, der afkoder dem og dermed har sine 5 kort, som A ikke kender. A vælger derefter 5 nye tilfældige kort til sig selv, indkoder dem en gang til med

$$c_{BiAi} = c_i' = E(c_i, e_A)$$

Resultatet sendes til B, der delvist afkoder det med

$$c_{Ai} = c_i'' = D(c_i', d_B)$$

Dette nye resultat sendes så tilbage til A der endeligt kan afkode det med

$$m_i = D(c_i'', d_A)$$

På samme måde fortsætter spillet sålænge spillerne vil have nye kort. Når spillet er slut, offentliggør spillerne deres nøgler, så de gensidigt kan kontrollere, at der ikke er blevet snydt.

Protokollen beskrevet ovenfor kan på samme måde bruges til andre former for kortspil, da den kun beskæftiger sig med en fair uddeling af kortene.

# Kapitel 8

# RSA-kryptosystemet

RSA-kryptosystemet er opkaldt efter Rivest, Shamir og Adleman, som opfandt og beskrev det første gang i 1977 [44].

## 8.1 Beskrivelse af RSA

I RSA benyttes følgende betegnelser :

| | |
|---|---|
| Klartekst | $m$ |
| Ciffertekst | $c$ |
| Indkodningsnøgle (offentlig) | $e$ |
| Afkodningsnøgle (hemmelig) | $d$ |
| Store primtal | $p, q$ |
| Fælles nøgle | $n = p \cdot q$ |
| Eulers $\phi$-funktion | $\phi(n) = (p-1)(q-1)$ |
| Offentlig nøgle | $e$ |
| Hemmelig nøgle | $d$ |

Når man skal henholdsvis ind- og afkode en meddelelse med RSA-systemet foregår det med følgende smæklås-funktion :

$$\begin{array}{lrcl} \text{Offentlig indkodningsfunktion} & E(m) & \equiv & m^e \bmod n \\ \text{Hemmelig afkodningsfunktion} & D(c) & \equiv & c^d \bmod n \end{array} \tag{8.1}$$

$D$ hemmeligholdes ved at skjule parametrene $d, p, q$. $E$ offentliggøres ved at lade $e$ og $n$ være kendt.

Ved indkodning repræsenteres klarteksten som et stort heltal i intervallet $[0, n-1]$. En lang meddelelse skal derfor deles op i blokke, som

hver især kan repræsenteres ved et sådant heltal. Ciffertekst en findes ved at beregne $c = E(m)$, og afkodes ved $m = D(c)$. Både klartekst- og ciffertekstblokke vil derfor ligge i intervallet $[0, n-1]$.

Ubrydeligheden af RSA-systemet hænger på at $p$ og $q$ er *store* primtal. Man kan som vi skrev i kapitel 4 godt teste om store tal (i størrelsesordenen 100 cifre) er primtal, men man kan ikke indenfor overskuelig tid faktorisere $n$, et tal i størrelsesordenen 200 cifre, således at $p$ og $q$ kan findes.

## 8.2 Valg af RSA parametre

For at kunne benytte de to algoritmer $E(m)$ og $D(c)$, skal tallene $p, q, e, d$ og $n$ fastlægges.

- Først findes to *store* primtal $p$ og $q$ med hver cirka 100 cifre, hvorefter $n$ beregnes som $n = p \cdot q$.
- Derefter vælges et $d$ så $\mathrm{SFD}(d, \phi(n)) = 1$ og $d > \max(p, q)$.
- Endelig beregnes $e$ som $d$'s multiplikative inverse modulo $\phi(n)$ så

$$e \cdot d \equiv 1 \bmod \phi(n)$$

### 8.2.1 Beregning af $p$ og $q$

Primtallene $p$ og $q$ kan findes ved, at vælge et ulige tilfældigt tal af den ønskede størrelsesorden og teste om det er et primtal. Sålænge dette ikke er tilfældet adderes 2, og tallet testes igen.

Det er vigtigt at $p$ og $q$ er forskellige, idet der ellers sker følgende :

Hvis $p$ er lig $q$, bliver $n = p^2$. Forsøger man så at indkode et $m$, der er et multiplum af $p$, dvs $(m = k \cdot p)$, med en eksponent $e$ på 3 eller mere fås :

$$\begin{aligned} m^e &= (kp)^e \\ &= k^e \cdot p^e \\ &= k^e \cdot p^{(2+e-2)} \\ &= k^e \cdot p^2 \cdot p^{e-2} \qquad \Rightarrow \\ m^e &\equiv k^e \cdot 0 \cdot p^{e-2} \bmod p^2 \qquad \Rightarrow \\ m^e &\equiv 0 \bmod p^2 \end{aligned}$$

Alle $m = kp$ indkodes altså til nul, og kan derfor ikke genskabes ved afkodning.

### 8.2.2 Betingelser til nøglerne $e$ og $d$

RSA's funktion afhænger først og fremmest af, at ind- og afkodningsalgoritmerne $E(m)$ og $D(c)$ er hinandens inverse. Det vil altså sige at :

$$m = D(E(m)) = (m^e \text{ MOD } n)^d \text{ MOD } n$$

Dette er formaliseret i følgende sætning :

**Sætning 8.1** *Hvis*

$$n = p \cdot q, \text{ hvor } p \neq q$$

*og hvis d og e opfylder*

$$e \cdot d \equiv 1 \bmod \phi(n) \tag{8.2}$$

*gælder det for alle* $m \in [0, n-1]$ *at :*

$$(m^e \text{ MOD } n)^d \text{ MOD } n = m$$

**Bevis :**

Idet $n = p \cdot q$, bliver

$$\phi(n) = (p-1) \cdot (q-1)$$

og vi har at

$$(m^e \text{ MOD } n)^d \text{ MOD } n = m^{ed} \text{ MOD } n$$

Fra Fermats sætning 2.22 ved vi at når $\text{SFD}(m,p) = 1$, er

$$\begin{array}{lcll} m^{p-1} & \equiv & 1 \bmod p & \Rightarrow \\ m^{(p-1)(q-1)} & \equiv & 1^{q-1} \bmod p & \Rightarrow \\ m^{\phi(n)} & \equiv & 1 \bmod p & \Rightarrow \\ m^{k\cdot\phi(n)} & \equiv & 1 \bmod p & \Rightarrow \\ m^{k\cdot\phi(n)} \cdot m & \equiv & m \bmod p & \Rightarrow \\ m^{k\cdot\phi(n)+1} & \equiv & m \bmod p & \end{array} \tag{8.3}$$

Hvis $\text{SFD}(m,p) > 1$ må $m$ være et multiplum af $p$, da $p$ er et primtal. Vi har derfor at

$$\text{SFD}(m,p) > 1 \Rightarrow m \equiv 0 \bmod p$$

Man ser, at for sådanne $m$ gælder sidste linie i formel 8.3 også. Den gælder altså for *alle* $m$. En fuldstændig tilsvarende argumentation kan gennemføres for $m$ og $q$, så man får

$$m^{k\cdot\phi(n)+1} \equiv m \bmod q \tag{8.4}$$

Da $\mathrm{SFD}(p,q) = 1$ kan vi ud fra formlerne 8.3 og 8.4 og sætning 2.19 slutte, at det for alle $m$ gælder

$$m^{k\cdot\phi(n)+1} \equiv m \bmod pq$$

hvilket leder os frem til at

$$m^{ed} \equiv m \bmod n$$

□

Hvis $\mathrm{SFD}(d,\phi(n)) = 1$ ved vi, at betingelsen på $e$ og $d$ kan opfyldes, da følgende ligning ifølge sætning 2.14 altid har en løsning :

$$d \cdot e \equiv 1 \bmod \phi(n) \tag{8.5}$$

og vi har i afsnit 3.2.2 vist, at løsningen kan findes $\mathcal{O}(\log^3 \phi(n))$.

### *Eksempel 8.1. RSA-kryptosystemet*

*Et eksempel med små tal kan illustrere princippet i RSA. Det skal dog understreges at selve ubrydeligheden af systemet ene og alene er afhængig af, at $p$ og $q$ er* store *primtal.*

*Vi vælger de følgende værdier til de variable :*

$$\begin{aligned} p &= 47 \\ q &= 67 \\ n &= p \cdot q = 47 \cdot 67 = 3149 \end{aligned}$$

*Så er*

- $\phi(3149) = 46 \cdot 66 = 3036$.
- $d$ *kan da vælges til* 1489.
- $e \equiv d^{-1} \bmod 3036$ *beregnes til* 157.

*Hvis vi repræsenterer hvert bogstav ved to cifre, kan vi med* $n = 3149$ *indkode to bogstaver pr blok :*

$$\begin{aligned} \textit{mellemrum} &= 00 \\ \texttt{A} &= 01 \\ \texttt{B} &= 02 \\ &\vdots \\ \texttt{A} &= 29 \end{aligned}$$

*Således vil klartekstmeddelelsen*

VEL ER JEG IKKE CÆSAR (Elverhøj, Kuhlau)

*transformeres til*

*2205 1200 0518 0010 0507 0009 1111 0500 0327 1901 1800*

*Den første blok* ($m = 2205$) *indkodes ved*

$$c = 2205^{157} \text{ MOD } 3149 = 1589$$

*Hele klartekst meddelelsen indkodes således til :*

*1589 0852 0894 2897 0440 1734 0352 2185 0091 2895 2621*

*Afkodningen af den første blok* $c = 1589$ *foregår på samme måde :*

$$m = 1589^{1489} \text{ MOD } 3149 = 2205$$

*Hele ciffertekst meddelelsen afkodes således til :*

*2205 1200 0518 0010 0507 0009 1111 0500 0327 1901 1800*

## 8.3 RSA og digital underskrift

Fordi RSA er et offentlignøgle system og fordi mængden af klartekster er lig mængden af ciffertekster kan RSA anvendes til digital underskrift. En beregning af en signeret og kodet meddelelse fra $A$ til $B$ vil da kunne gøres ved

$$\begin{aligned} c_s &= D(m, d_A, n_A) = m^{d_A} \text{ MOD } n_A \\ c_{sh} &= E(c_s, e_B, n_B) = c_s^{e_B} \text{ MOD } n_B \end{aligned}$$

hvor $c_s$ er en signeret, men ikke hemmeligholdt meddelelse og $c_{sh}$ er en signeret *og* hemmeligholdt meddelelse.

Hvis $n_A$ er større end $n_B$, kan der imidlertid opstå et problem. den signerede meddelelse $c_s$ ligger jo i intervallet $[0, n_A - 1]$, og kan derfor blive større end $n_B$. Det betyder at man ikke direkte kan lade signatur og hemmeligholdelse følge hinanden. Man er nødt til at ombryde den signerede meddelelse $c_s$ til nye blokke for at den med sikkerhed ligger i intervallet $[0, n_B - 1]$.

Problemet kan dog løses, uden at man behøver at ombryde meddelelsen. [44] foreslår at hver bruger udstyres med to sæt nøgler. Et $(n_s, e_s, d_s)$ til signatur, og et $(n_h, e_h, d_h)$ til hemmeligholdelse. For alle $n_s$ og $n_h$ skal der gælde at

$$n_s < t < n_h$$

for en generel tærskel værdi $t$. A sender således en hemmelig, signeret meddelelse til B ved

$$c_{sh} = E(D(m, d_{As}, n_{As}), e_{Bh}, n_{Bh})$$

Dette kan altid beregnes, da $n_{As}$ og $n_{Bh}$ holder sig på hver sin side af $t$. Afkodningen hos B foregår tilsvarende ved

$$m = E(D(c_{sh}, d_{Bh}, n_{Bh}), e_{As}, n_{As})$$

# Kapitel 9

# RSA-sikkerhed

I de 12 år RSA-systemet har været kendt, er det ikke lykkedes at bryde det. Der er imidlertid blevet foreslået en række angreb mod systemet, der viser at systemet har nogle svage punkter, hvis man ikke tænker sig om. Vi vil her vurdere RSA-systemets sikkerhed ved at beskrive de vigtigste af disse resultater.

## 9.1 Kryptoanalyse af RSA-systemet

Diskussionen i det følgende giver et overblik, over de muligheder man har for at bryde systemet. Det sker primært ved at beregne de funktioner og konstanter, der indgår i systemet :

- $p$ og $q$.
- $\phi(n)$.
- Nøglen $d$.
- Afkodningsfunktionen $D$.

De fleste af argumenterne i det følgende er taget fra [44]. Dog er det følgende afsnit fra [56].

### 9.1.1 Uheldige meddelelser

Hvis en kryptoanalytiker finder en meddelelse $m$ eller opsnapper en ciffertekst $c$, der ikke er indbyrdes primisk med modulussen $n$, kan systemet umiddelbart brydes. Dette ses af følgende argument :

Da $m$ er mindre end $n$, må

$$\mathrm{SFD}(m,n) < n$$

Da $n$ er $p \cdot q$ kan $\mathrm{SFD}(m,n)$ kun antage tre værdier, 1, $p$ eller $q$ Hvis vi ved at

$$\mathrm{SFD}(m,n) > 1$$

må resultatet derfor enten være $p$ eller $q$. Hvis kryptoanalytikeren får opsnappet en sådan meddelelse, kan hun bryde systemet, da den anden faktor i $n$ let kan beregnes ved $\frac{n}{q}$ eller $\frac{n}{p}$. Når hun har beregnet $p$ og $q$ kan $\phi(n)$ let beregnes og dermed også $d$, som $e$'s inverse modulo $\phi(n)$.

Heldigvis er sandsynligheden for at ramme en sådan uheldig meddelelse $m$, hvor $\mathrm{SFD}(m,n) > 1$ meget lille. Da antallet af meddelelser hvor $\mathrm{SFD}(m,n) = 1$ er $\phi(n)$, er antallet af ulovlige meddelelser lig $n - \phi(n)$. Derfor bliver sandsynligheden :

$$\begin{aligned}
\frac{n-\phi(n)}{n} &= \frac{pq-(p-1)(q-1)}{pq} \\
&= \frac{pq-pq+p+q-1}{pq} \\
&\approx \frac{p+q}{pq} \\
&= \frac{1}{p}+\frac{1}{q}
\end{aligned}$$

Da $p$ og $q$ begge er cirka 100 decimale cifre lange tal, får man brøkdelen af ulovlige meddelelser til :

$$\frac{1}{10^{100}}+\frac{1}{10^{100}} = 2 \cdot 10^{-100}$$

Selv hvis RSA-systemet bruges til at transmittere 1 milliard meddelelser pr sekund, vil universets levetid være opbrugt mange gange, før det er sandsynligt, at en sådan meddelelse eller ciffertekst opstår.

### 9.1.2 Faktorisering af $n$

Den mest umiddelbare måde at bryde RSA-systemet på, er ved at faktorisere $n$. Så får man straks $p$ og $q$, og kan beregne $\phi(n)$ og derved $d$.

Som vi har vist, har de bedste generelle faktoriseringsalgoritmer kompleksiteten

$$\mathcal{O}\left(e^{c\sqrt{\log n \log\log n}}\right)$$

Selv de til dato hurtigste og mest avancerede implementeringer må, som beskrevet i kapitel 4, give op, inden tallene når de 100 cifre. Den øvre grænse for faktorisering er gennem de sidste 10 år vokset med i gennemsnit 5 cifre pr år. Dette giver dog ingen garanti for, at der en dag kommer et gennembrud på dette område, hvorved kompleksiteten kan bringes længere ned. Det kan måske i et spring muliggøre faktorisering af langt større tal.

Selvom det ser ud til at RSA-systemet ikke kan brydes med en generel faktoriseringsalgoritme, skal man dog passe på ikke at drage for vidtrækkende konklusioner om dette. I forhold til en generel faktoriseringsalgoritme har man nemlig ekstra oplysninger om tallet $n$ og dets faktorer $p$ og $q$.

- $n$ har netop to primfaktorer.
- Man kender et $e$, hvor der gælder

$$e \cdot d \equiv 1 \bmod (p-1)(q-1)$$

- Man kan få adgang til par $(c, m)$, hvor der gælder

$$\begin{aligned} c^d &\equiv m \bmod n \text{ og} \\ m^e &\equiv c \bmod n \end{aligned}$$

- $p$ og $q$ er (som vi skal se senere) som regel en speciel type primtal.

Der er dog endnu ingen, der har kunnet udnytte disse ekstra oplysninger til at konstruere en specielt hurtig RSA-faktoriseringsmetode.

Faktoriseringen leder altså ikke til en brugbar metode til at bryde RSA-systemet. Det giver imidlertid en *øvre* grænse for, hvor svært det er at bryde systemet, da systemet altid kan brydes af denne vej.

### 9.1.3 Beregning af $\phi(n)$

Hvis man kan beregne $\phi(n)$ uden at faktorisere $n$, vil man også kunne bryde systemet, idet $n$ på baggrund af denne viden let kan faktoriseres, hvorved vi kan finde $p$ og $q$. Det vil foregår således :

Vi har

$$n = p \cdot q \Leftrightarrow p = \frac{n}{q}$$

Vi har ligeledes at :

$$\begin{aligned}
\phi(n) &= (p-1)(q-1) \\
&= pq - p - q + 1 \\
&= n - p - q + 1 \\
&= n - \frac{n}{q} - q + 1 \Leftrightarrow \\
\frac{n}{q} + q &= n - \phi(n) + 1 \Leftrightarrow \\
n + q^2 &= q\,(n - \phi(n) + 1) \Leftrightarrow \\
q &= \frac{n - \phi(n) + 1 \pm \sqrt{(n - \phi(n) + 1)^2 - 4n}}{2} \qquad (9.1)
\end{aligned}$$

$p$ kan beregnes på helt tilsvarende måde.

Beregning af $\phi(n)$ giver altså en metode til faktorisering af $n$. Foreløbig har ingen dog kunnet udnytte dette til at lave en hurtig faktoriseringsmetode. Blandt forsøgene er [23] der ud fra det faktum,at der for alle primtal $p$ gælder

$$p^2 \equiv 1 \bmod 24$$

har konstrueret en formel til beregning af $\phi(n)$ hvor :

$$\phi(n) = \frac{24 \cdot x}{n - 1}$$

Den eneste ubekendte i ovenstående ligning er $x$. Men da der ikke vides andet om $x$, end at den er et heltal af størrelsesordenen $n^2$ (idet $\phi(n)$ er af størrelsesordenen $n$), løber det ud i sandet.

Det er ikke tilstrækkeligt at hemmeligholde $p, q$ og $\phi(n)$, da sidstnævnte kan beregnes, hvis man kender enten summen $p+q$ eller differensen $p-q$. Disse må derfor heller ikke offentliggøres.

Kender man $(p+q)$, findes $\phi(n)$ som følger :

$$\phi(n) = (p-1)\cdot(q-1) = p\cdot q - (p+q) + 1 \tag{9.2}$$

Har man $(p-q)$, omskrives formel 9.2 og man får :

$$\begin{aligned}
\phi(n) &= (n+1) - (p+q) \\
&= (n+1) - \sqrt{(p+q)^2} \\
&= (n+1) - \sqrt{p^2+q^2+2pq} \\
&= (n+1) - \sqrt{p^2+q^2-2pq+4pq} \\
&= (n+1) - \sqrt{(p-q)^2+4n}
\end{aligned}$$

### 9.1.4 Beregning af $d$

Hvis man kunne beregne $d$ uden først at beregne $p, q$ eller $\phi(n)$, så kan systemet også brydes, da man så får kendskab til den hemmelige afkodningsnøgle.

Kender man $d$, kendes imidlertid også et multiplum af $\phi(n)$ da

$$e\cdot d \equiv 1 \bmod \phi(n) \Rightarrow e\cdot d - 1 = k\cdot\phi(n)$$

[33, side 144] viser, at i så fald kan $n$ faktorisere effektivt. Har man derfor en metode til at beregne $d$, har man også en metode til at faktorisere $n$.

Systemet ville også være brudt hvis man kunne finde et $d' \neq d$, hvor der gælder :

$$e\cdot d' \equiv 1 \bmod \phi(n)$$

Dette $d'$ ville nemlig også kunne bruges som afkodningsnøgle. En måde at finde et sådant $d'$ på, ville være at lede efter det ved at afprøve alle tal fra 1 og op. Det følgende argument, viser imidlertid at sådanne $d'$ mindst vil være af smme størrelsesorden som det oprindelige $d$, hvorfor denne metode ikke er særlig hensigtsmæssig.

Hvis man imidlertid har et sådant $d'$, vil der gælde at :

$$e\cdot d' - 1 = k'\cdot\phi(n)$$

ligesom for $e$ og det originale $d$

$$e \cdot d - 1 = k \cdot \phi(n)$$

Man får derfor, ved at trække de to ligninger fra hinanden

$$(e \cdot d' - 1) - (e \cdot d - 1) = k' \cdot \phi(n) - k \cdot \phi(n)$$

som giver

$$e \cdot (d' - d) = (k' - k) \cdot \phi(n)$$

Da $e$ er indbyrdes primisk med $\phi(n)$, må forskellen mellem de forskellige $d'$ være et multipla af $\phi(n)$. Hvis $d$ er af størrelsesorden $\phi(n)/2$, vil $d'$ mindst være af samme størrelsorden. Derfor kan det ikke betale sig at søge efter $d'$.

### 9.1.5 Beregning af afkodningsalgoritmen

En sidste mulighed for at bryde RSA-kryptosystemet er ved afkode cifferteksterne uden overhovedet at beregne $d$, $\phi(n)$ eller $p$ og $q$. Det kunne f.eks. gøres ved at beregne *den e'te rod modulus n* af ciffertekst $c$. Den er jo lavet som

$$\begin{aligned} c &\equiv m^e \bmod n \\ m &\equiv c^d \bmod n \end{aligned}$$

Bemærk at der ikke er tale om at tage den normale rod $\sqrt[e]{c}$, men om at finde det heltal $m$ som opløftet i $e$'te potens giver $c$ regnet modulus $n$.

Rivest, Shamir og Adleman [44] giver ikke noget argument for at beregning af den $e$'te rod modulo $n$ ikke kan gøres effektivt, men da vi ikke er stødt på nogen, der giver en algoritme til at gøre det, må det være et tegn på, at det ikke er praktisk beregneligt.

### 9.1.6 Opsamling

Gennemgangen i dette afsnit har vist at alle de beskrevne metoder, bortset fra den sidste, beregning af afkodningsalgoritmen, giver en metode til faktorisering af $n$. Det betyder, at det er lige så svært at finde afkodnings*nøglen* som at faktorisere $n$. Det viser imidlertid også, at det *måske* er muligt at afkode meddelelserne uden at faktorisere $n$.

Hvis man kunne vise, at alle mulige metoder til at bryde RSA-systemet, vil resultere i en metode til faktorisering af $n$, så vill man have vist, at sikkerheden af RSA-systemet er *lig* faktorisering. Dette ville det være et gennembrud for systemet. Dels vil man have begrænset alle kryptoanalytiske angreb til at koncentrere sig om et bestemt fagområde, og dels vil man vide præcis hvor sikkert systemet er, som en funktion af nøglestørrelsen. Derved ville man i et konkret system kunne lægge sig fast på et bestemt sikkerhedsniveau, ved at vælge en bestemt nøglestørrelse.

## 9.2 Valg af $p$ og $q$

Efter at have givet et overblik over angreb på RSA-systemet, vil vi i dette og det følgende afsnit, vise hvilke $p, q, e$ og $d$ man bør vælge for at gøre systemet så sikkert som muligt.

Vi har allerede nævnt, at $p$ og $q$ bør være forskellige, meget store samt at deres sum og differens ikke må offentliggøres, for at systemet er sikkert.

Derudover bør $p$ og $q$ være af samme størrelsesorden. Mange faktoriseringsalgoritmer har nemlig en kompleksitet, der afhænger af den *mindste* af faktorerne. Samtidig må de ikke ligge for tæt på hinanden. Gør de det kan en sekventiel afprøvning startende ved $\sqrt{n}$ give et resultat. I eksempel 8.1 kunne vi have fundet $q$ meget hurtigt ved først at gætte på

$$q = \lfloor\sqrt{3149}\rfloor + 1 = 57$$

for derefter at prøve med tallene 59, 61, 63, 65 og tilsidst 67, som er det rigtige tal. På hvert trin tester man tallet ved at undersøge, om det går op i $n$. Er det tilfældet, har man fundet $p$ eller $q$, og systemet er brudt.

Ligesom nogle faktoriseringsalgoritmers kompleksitet afhænger af det sammensatte tals mindste faktor, findes der ifølge [44] og [56] faktoriseringsalgoritmer, som er gode, hvis det for en faktor $p$ gælder at $p-1$ eller $p+1$ har små primfaktorer. For RSA-systemet betyder dette, at $p-1$, $p+1$, $q-1$ og $q+1$ skal konstrueres således, at de har så store primfaktorer som muligt. Det er klart at 2 er en faktor i alle fire tal, da både $p$ og $q$ er ulige, men de øvrige faktorer i tallet kan jo godt være store.

Ser vi på eksemplet fra afsnit 8.1, har vi

$$\begin{array}{rcrcrcl} p & = & 47 & p-1 & = & 46 & = & 2 \cdot 23 \\ & & & p+1 & = & 48 & = & 2^4 \cdot 3 \\ \\ q & = & 67 & q-1 & = & 66 & = & 2 \cdot 3 \cdot 11 \\ & & & q+1 & = & 68 & = & 2^2 \cdot 17 \end{array}$$

Vi ser at $p$ og $q$ er valgt, så kun $p-1$ er "god", mens både $p+1$, $q-1$ samt $q+1$ har små faktorer. Tallene fra dette eksempel opfylder altså ikke de opstillede krav og må derfor betragtes, som "svage" primtal.

### 9.2.1 Stærke primtal

De ovenfor beskrevne krav på $p$ og $q$ har ført til, at man har indført begrebet **et stærkt primtal**. Der findes flere forskellige definitioner på begrebet. Den vi har valgt, stammer fra [22], og er den stærkeste forstået på den måde, at den kræver mest af $p$ og $q$.

**Definition 9.1 (Stærke primtal)**
*Et primtal p kaldes et stærk primtal, hvis det opfylder følgende :*

$p-1$ *har en stor primfaktor* $r$

$p+1$ *har en stor primfaktor* $s$

$r-1$ *har en stor primfaktor* $t$

*Hvilket betyder :*

$$p \equiv \begin{cases} 1 \mod r \\ -1 \mod s \end{cases} \tag{9.3}$$

*og*

$$r \equiv 1 \bmod t \tag{9.4}$$

Betingelsen på $r$ er ikke begrundet af den ovenstående diskussion. Årsagen vil derimod fremgå af afsnittet om dekryptering ved gentagen indkodning.

### Hvordan konstrueres stærke primtal

Ideen er, at man starter med at udvælge $s$ og $t$ og ud fra disse konstruerer $r$ og $p$, således at betingelserne er opfyldt.

1. $s$ vælges som det mindste primtal større end et tilfældigt tal af den ønskede størrelse. $t$ vælges på samme måde.

2. For at opfylde formel 9.3 skal $r$ være på formen $r = 2lt + 1$. Faktoren 2 skyldes at vi kun behøver at lede blandt de ulige tal. Vi løber derfor tal af formen $2lt + 1$ igennem for $l = 1, 2, 3, \ldots$ indtil den ønskede størrelse er nået, hvorefter vi primtalstester hvert efterfølgende kandidat til $r$ indtil testen giver et positivt resultat.

3. Da
$$p \equiv 1 \bmod r \quad \text{og} \quad p \equiv -1 \bmod s$$
kan $p$ findes ved at benytte den kinesiske restsætning. Vi finder først $s^{-1} \bmod r$ og $r^{-1} \bmod s$ :
$$\left.\begin{array}{rcl} s^{r-1} & \equiv & 1 \bmod r \\ s^{-1}s & \equiv & 1 \bmod r \end{array}\right\} \Leftrightarrow \quad s^{-1} \equiv s^{r-2} \bmod r$$
$$\left.\begin{array}{rcl} r^{s-1} & \equiv & 1 \bmod s \\ r^{-1}r & \equiv & 1 \bmod s \end{array}\right\} \Leftrightarrow \quad r^{-1} \equiv r^{s-2} \bmod r$$
Med koefficienterne 1 og $-1$ giver den kinesiske restsætning at
$$\begin{aligned} p &\equiv 1 \cdot s^{r-2}s - 1 \cdot r^{s-2}r \bmod rs \\ &\equiv s^{r-1} - r^{s-1} \bmod rs \end{aligned}$$
Den principale løsning til ligning 9.3 benævner vi
$$u(r,s) = s^{r-1} - r^{s-1} \tag{9.5}$$
Da $p$ skal være ulige, kan den skrives på formen
$$p = p_0 + 2krs$$
hvor $p_0$ er :
$$p_0 = \begin{cases} u(r,s) & \text{hvis } u(r,s) \text{ ulige} \\ u(r,s) + rs & \text{hvis } u(r,s) \text{ lige.} \end{cases} \tag{9.6}$$

Vi behøver derfor kun at gennemsøge tal af formen $(p_0 + 2krs)$ for $k = 1, 2, 3, \ldots$ indtil den ønskede størrelse er nået og derpå primtalsteste $p$.

**Sætning 9.1** *Hvis $r$ og $s$ er ulige primtal, så opfylder $p$ ligning 9.3 hvis og kun hvis $p = p_0 + 2krs$, hvor $p_0$ er defineret som i ligning 9.6.*

**Bevis :**

Først viser vi, at tal på formen $p_0 + 2krs$ kan opfylde ligningen.

Ifølge Fermat har vi at

$$\begin{aligned} s^{r-1} &\equiv 1 \bmod r \text{ og at} \\ r^{s-1} &\equiv 1 \bmod s \end{aligned}$$

Det gælder også at

$$\begin{aligned} s^{r-1} &\equiv 0 \bmod s \text{ og} \\ r^{s-1} &\equiv 0 \bmod r \end{aligned}$$

Endelig er

$$\begin{aligned} rs &\equiv 0 \bmod r \text{ og} \\ rs &\equiv 0 \bmod s \end{aligned}$$

Derfor vil

$$\begin{aligned} u(r,s) &\equiv 1 \bmod r \\ u(r,s) &\equiv -1 \bmod s \end{aligned}$$

Dernæst viser vi, at tal, der *ikke* er på formen $p_0 + 2krs$, *ikke* kan opfylde ligning 9.3.

Antag at både $u$ og $u'$ opfylder 9.3 og betragt forskellen

$$u - u' = \begin{cases} (1 \text{ MOD } r) - (1 \text{ MOD } r) = 0 \text{ MOD } r = kr \\ (-1 \text{ MOD } s) - (-1 \text{ MOD } s) = 0 \text{ MOD } s = k's \end{cases}$$

hvor $k$ og $k'$ er konstanter.

Vi kan nu se at forskellen skal være et multiplum af $rs$. Derfor må både $u$ og $u'$ være på formen $u(r,s) + krs$. □

### Kompleksitet

Vi ved fra afsnit 2.2.3, at den gennemsnitlige afstand mellem primtallene er $\log(x)$, men da vi kun tester på de ulige tal større end $a$, behøver vi kun undersøge $\frac{\log(x)}{2}$ tal. Har $x$ $v$ cifre, er $x \approx 2^v$ og vi får

$$\begin{aligned} \frac{\log(2^v)}{2} &= \frac{v\log(2)}{2} \\ &\approx 0.35v \end{aligned}$$

tal, som skal undersøges inden vi finder $s$.

Kompleksiteten af at teste, om et heltal er et primtal, er $\mathcal{O}(v^3)$ ved brug af Miller og Rabins metode (se kapitel 4). Derfor vil kompleksiteten af at finde et *konkret* primtal med $v$ cifre være

$$\mathcal{O}(0.35v \cdot v^3) = \mathcal{O}(v^4) \approx c \cdot v^4$$

Kompleksiteten af at konstruere $r$ vil være den samme. Hvis $p$ er af størrelsesorden $2^v$ og $r, s, t$ af $2^{v/2}$, vil den samlede kompleksitet af at konstruere $p$ være :

$$\begin{aligned} 3 \cdot c \cdot \left(\frac{v}{2}\right)^4 + c \cdot (v^4) &= 3 \cdot c \cdot \left(\frac{v^4}{16}\right) + c \cdot (v^4) \\ &= \left(1 + \frac{3}{16}\right) \cdot c \cdot (v^4) \\ &= 1.19 \cdot (c \cdot v^4) \end{aligned}$$

Det er altså kun 19 % sværere at finde et stærkt primtal, end at finde et vilkårligt primtal.

I det følgende vil vi kalde de primtal, hvor $(p-1)$ og $(p+1)$ har færrest små divisorer, for **ekstra stærke primtal**. Det er tal, hvor hvor $l$ og $k$ er lig 1 således, at $p = 2r + 1$ og $p = 2s - 1$. Denne ekstra betingelse medfører imidlertid, at der er langt færre tal at vælge imellem. Man kan derfor forudse, at sådanne tal er sværere at finde.

## 9.3 Betingelser på $e$ og $d$

I forrige kapitel viste vi at

$$e \cdot d \equiv 1 \bmod \phi(n)$$

Hvis vi skal gøre RSA-systemet så sikkert som muligt, er det ikke kun til $p$ og $q$, der bør stilles skærpede krav. Også ind- og afkodningsnøglerne $e$ og $d$, skal opfylde visse betingelser.

Først og fremmest skal $e$ og $d$ være forskellige. Er de ens er ind- og afkodningsfunktionerne identiske, og der er ikke længere tale om et offentlig-nøgle system. Det betyder med andre ord, at enhver kan afkode en ciffertekstmeddelelse, ved at gentage indkodningen.

Derudover bør $d$ vælges mindst så stor som $p$ og $q$, for at sikre at en ciffertekst $c$ ikke i løbet af kort tid kan afkodes ved at beregne $c^2$ MOD $n, c^3$ MOD $n, \ldots$ indtil der fremkommer en meningsfyldt meddelelse, hvorved man har beregnet $c^d$ mod $n$ og afkodet meddelelsen.

Et tilsvarende krav gælder ikke for den offentlige nøgle $e$, da det at tage den e'te rod modulus $n$ af $c$, er en anderledes svær operation uanset størelsen af $e$, som beskrevet i afsnit 9.1.5. $e$ bør dog vælges så stor, at $m^e$ bliver større end $n$ for de fleste meddelelser så man ikke kun får lavet en potensopløftning, men også får beregnet modulus En meddelelse hvor dette ikke sker, kan let afkodes ved at beregne den $e$'te rod af meddelelsen - en almindelig simpel regneoperation. Dette stiller ikke noget voldsomt krav til $e$. Hvis vi i vores eksempelsystem med $n = 3149$ vælger $e$ til 3, er det kun meddelelser hvis tal-repræsentation mindre end 15, der ikke får en modulus større end 0, idet $15^3 = 3375$ er større end $n$.

En tredje vigtig betingelse på den offentlige nøgle $e$ er, at den skjuler meddelelserne.

$$m^e \text{ MOD } n$$

må altså ikke være den identiske afbildning.

### *Eksempel 9.1. Identisk indkodning med RSA*

*Et eksempel på ovenstående er RSA-kryptosystemet med de følgende parametre :*

$$\begin{array}{lllll} p & = & 97 & q & = & 109 \\ n & = & 97 \cdot 109 & & = & 10573 \\ e & = & 865 & d & = & 9505 \end{array}$$

*I dette eksempel vil indkodning af alle meddelelser $m$ i intervallet $[0, n-1]$ resultere i den identiske afbildning. Dette system*

*giver altså overhovedet ingen hemmeligholdelse. F.eks indkodes meddelelsen 2314 til :*

$$2314^{865} \text{ MOD } 10573 = 2314$$

Følgende sætning viser hvor mange meddelelser, der indkodes til sig selv.

**Sætning 9.2** *Antallet $A(n)$ af meddelelser, der indkodes til sig selv, i et givet RSA-system er givet ved*

$$A(n) = (1 + \text{SFD}(e-1, p-1))(1 + \text{SFD}(e-1, q-1)) \tag{9.7}$$

**Bevis :**

Vi starter med at vise at kongruensen

$$x^e \equiv x \bmod p \tag{9.8}$$

hvor $p$ er et primtal, har $(1 + \text{SFD}(e-1, p-1))$ løsninger.

Hvis $g$ er en frembringer, og $x = g^j$, kan formel 9.8 omformes til

$$g^{je} \equiv g^j \bmod p$$

Fra sætning 2.23 ved vi, at da $\text{SFD}(g, p)$ er 1, kan det omskrives til

$$\begin{aligned} j \cdot e &\equiv j \bmod \phi(p) \Rightarrow \\ j \cdot (e-1) &\equiv 0 \bmod p-1 \end{aligned} \tag{9.9}$$

Divideres resultatet med $\text{SFD}(e-1, p-1)$, fås

$$j \cdot \frac{e-1}{\text{SFD}(e-1, p-1)} \equiv 0 \bmod \frac{p-1}{\text{SFD}(e-1, p-1)}$$

Nu er

$$\frac{(e-1)}{\text{SFD}(e-1, p-1)} \text{ og } \frac{(p-1)}{\text{SFD}(e-1, p-1)}$$

indbyrdes primiske, så

$$\frac{(e-1)}{\text{SFD}(e-1, p-1)}$$

kan divideres ud :

$$j \equiv 0 \bmod \frac{p-1}{\mathrm{SFD}(e-1,p-1)}$$

Heraf ses det, at ligningen 9.9 har løsningerne

$$j = k \cdot \frac{p-1}{\mathrm{SFD}(e-1,p-1)} \text{ hvor } k = 0,1,\ldots,\mathrm{SFD}(e-1,p-1)-1$$

Endelig er $j = p-1$ også en løsning til ligning 9.8, hvorfor der i alt er

$$1 + \mathrm{SFD}(e-1,p-1)$$

løsninger til ligningen.

Den tilsvarende ligning i $q$ har af samme grund

$$1 + \mathrm{SFD}(e-1,q-1)$$

løsninger. Vi kan ud fra sætning 2.25 konkludere at antallet af løsninger ialt er :

$$(1 + \mathrm{SFD}(e-1,p-1))(1 + \mathrm{SFD}(e-1,p-1))$$

□

Da vi ved at $e$, $p$ og $q$ er ulige, må

$$\mathrm{SFD}(e-1,p-1) \text{ og } \mathrm{SFD}(e-1,q-1)$$

begge mindst være 2. Sætningen siger så at der derfor mindst er

$$(1+2)\cdot(1+2) = 9$$

meddelelser der indkodes til sig selv. Videre ser man at dette minimum kun nås, hvis der om $e$ gælder :

$$\mathrm{SFD}(e-1,p-1) = 2 \text{ og } \mathrm{SFD}(e-1,q-1) = 2$$

Hermed har vi skaffet os en præcis betingelse på, hvordan $e$ skal se ud.

Hvis $p$ og $q$ er stærke primtal :

$$p = 2la + 1 \text{ og } q = 2kb + 1$$

betyder ovenstående betingelse :

$$\begin{aligned} \mathrm{SFD}(e-1,p-1) &= 2 \Rightarrow \\ \mathrm{SFD}(e-1,2la) &= 2 \Rightarrow \\ \mathrm{SFD}(e-1,la) &= 1 \Rightarrow \\ \mathrm{SFD}(e-1,l) &= 1 \text{ og } \mathrm{SFD}(e-1,a)=1 \end{aligned}$$

For $q$ fås på samme måde

$$\mathrm{SFD}(e-1,k)=1 \text{ og } \mathrm{SFD}(e-1,b)=1$$

Fra forrige kapitel om RSA-systemet ved vi, at

$$\begin{aligned} &\mathrm{SFD}(e,\phi(n))=1 \Rightarrow \\ &\qquad \mathrm{SFD}(e,p-1) = 1 \text{ og } \mathrm{SFD}(e,q-1)=1 \end{aligned}$$

Med en udledning som ovenfor, får vi yderligere 5 betingelser på $e$, og i alt har vi at

$$\begin{array}{llll} \mathrm{SFD}(e,a) = 1 & & \mathrm{SFD}(e,b) = 1 \\ \mathrm{SFD}(e-1,a) = 1 & & \mathrm{SFD}(e-1,b) = 1 \\ \mathrm{SFD}(e,l) = 1 & & \mathrm{SFD}(e,k) = 1 \\ \mathrm{SFD}(e-1,l) = 1 & & \mathrm{SFD}(e-1,k) = 1 \\ \mathrm{SFD}(e,2) = 1 \end{array}$$

Hvis vi vender tilbage til eksemplet fra før med

$$\begin{array}{llll} p = 97 & & q = 109 \\ n = 97 \cdot 109 & & = 10573 \\ e = 865 & & d = 9505 \end{array}$$

får vi

$$\mathrm{SFD}(e-1,p-1)=\mathrm{SFD}(864,96)=96$$

$$\mathrm{SFD}(e-1,q-1)=\mathrm{SFD}(864,108)=108$$

og antallet af meddelelser, der indkodes til sig selv, bliver

$$(96+1)\cdot(108+1)=10573=n$$

hvilket betyder, at alle meddelelser indkodes til sig selv, præcis som vi postulerede.

## 9.4 Gentagen indkodning

Hvis det ikke lykkes for en kryptoanalytiker, at bryde RSA-systemet generelt, dvs finde den hemmelige indkodningsnøgle $d$, beregne $\phi(n)$ og finde $p$ og $q$, kan der godt tænkes at være huller i systemet, som tillader, at nogle ciffertekster kan afkodes uden kendskab til den hemmelige nøgle. Dette ville være en meget uheldigt egenskab ved RSA-systemet, da et offentlig-nøgle system hvor cifferteksterne kan afkodes uden brug af den hemmelige nøgle, jo ikke er meget værd. En måde at gøre det på, kunne være at finde en klartekst ved gentagen indkodning af cifferteksten.

Ved gentagen indkodning af en ciffertekst beregnes :

$$E(c),\ E(E(c)),\ E(E(E(c))),\ E(E(E(E(c)))),\ \ldots$$

Hvis klarteksten skal fremkomme ved gentagen indkodning skal

$$\begin{array}{lcll} E(c) & = & m & \text{eller} \\ E(E(c)) & = & m & \text{eller} \\ E(E(E(c))) & = & m & \text{eller} \\ E(E(E(E(c)))) & = & m & \text{eller} \\ E(E(\ldots(c)\ldots) & = & m & \end{array}$$

I RSA-systemet skal vi altså forsøge at finde klartekstmeddelelsen ved at beregne

$$\begin{array}{rcccll} c^e \text{ MOD } n & = & & m^{e^2} \text{ MOD } n = m & \text{eller} \\ (c^e)^e \text{ MOD } n & = & c^{e^2} \text{ MOD } n & = & m^{e^3} \text{ MOD } n = m & \text{eller} \\ ((c^e)^e)^e \text{ MOD } n & = & c^{e^3} \text{ MOD } n & = & m^{e^4} \text{ MOD } n = m & \text{eller} \\ (((c^e)^e)^e)^e \text{ MOD } n & = & c^{e^4} \text{ MOD } n & = & m^{e^5} \text{ MOD } n = m & \text{eller} \\ & & \vdots & & & \end{array}$$

Vi vil i den følgende beskrive mulighederne for gentagen indkodning med RSA-systemet.

**Sætning 9.3** *Hvis*

$$e^u = k \cdot \text{ORD}(m, n) + 1$$

*vil man ved u gentagne indkodninger få klarteksten frem.*

**Bevis :**

$$\begin{aligned} m^{(e^u)} &= m^{k \cdot \mathrm{ORD}_{(m,n)}+1} \text{ MOD } n \\ &= m \cdot m^{k \cdot \mathrm{ORD}_{(m,n)}} \text{ MOD } n \\ &= m \cdot (m^{\mathrm{ORD}_{(m,n)}})^k \text{ MOD } n \\ &= m \cdot 1^k \text{ MOD } n \\ &= m \text{ MOD } n \end{aligned}$$

□

Hvis vi skal analysere mulighederne for at bryde RSA-systemet ved gentagen indkodning, må vi se på hvor stor $u$ er. Vi starter med at se på $m$'s orden i $n$.

Vi har tidligere vist at $m$'s orden går op i $n$'s orden. Da $n$'s orden er $\phi(n) = (p-1)(q-1)$, betyder dette at $m$'s orden skal gå op i $\phi(n)$. Vælger vi $p$ og $q$ som stærke primtal :

$$p = 2la + 1 \text{ og } q = 2kb + 1$$

får vi

$$\phi(n) = (2la)(2kb) = 4lkab$$

$m$'s orden skal derfor være en af de følgende :

$$1, 2, 4, \text{alle divisorer i } k \text{ og } l \text{ }, a, b, 2a, 2b, \ldots, ab, \ldots, 4ab$$

Vælges $p$ og $q$ som ekstra stærke primtal, hvor $l$ og $k$ begge er en, må ordenen være en af :

$$1, 2, 4, a, b, 2a, 2b, \ldots, ab, \ldots, 4ab$$

[2] har vist følgende sætning om $m$'s orden i $n$ :

**Sætning 9.4** *For to ekstra stærke primtal p og q hvor :*

$$p = 2a + 1 \text{ og } q = 2b + 1$$

| *Orden* | *Antal elementer* | *m ligger i* |
|---|---|---|
| 1 | 4 | $(A(p) \cap A(q))$ |
| 2 | 5 | $(A(p) \cap B(q))$ |
| | | $\cup(B(p) \cap A(q))$ |
| | | $\cup(B(p) \cap B(q))$ |
| $a$ | $2(a-1)$ | $(A(q) \cap C(p))$ |
| $b$ | $2(b-1)$ | $(A(p) \cap C(q))$ |
| $2a$ | $4(a-1)$ | $(B(q) \cap C(p))$ |
| | | $\cup(B(q) \cap D(p))$ |
| | | $\cup(A(q) \cap D(p))$ |
| $2b$ | $4(b-1)$ | $(B(p) \cap C(q))$ |
| | | $\cup(B(p) \cap D(q))$ |
| | | $\cup(A(p) \cap D(q))$ |
| $ab$ | $(a-1)(b-1)$ | $(C(p) \cap C(q))$ |
| $2ab$ | $3(a-1)(b-1)$ | $(C(p) \cap D(q))$ |
| | | $\cup(C(q) \cap D(p))$ |
| | | $\cup(D(p) \cap D(q))$ |

**Tabel 9.1:** $m$'s orden i $n$

*gælder der, at m's orden i n, fordeler sig som vist i tabel 9.1, hvor*

$$A(p) \ : \ \{m | m \equiv 0 \bmod p \vee m \equiv 1 \bmod p\}$$

$$B(p) \ : \ \{m | m \equiv -1 \bmod p\}$$

$$C(p) \ : \ \{u | u \equiv m^2 \bmod p \wedge m \notin A(p) \wedge m \notin B(p)\}$$

$$D(p) \ : \ \{u | u \equiv -m^2 \bmod p \wedge m \notin A(p) \wedge m \notin B(p)\}$$

Sætningen viser at praktisk talt alle meddelelser vil have en periode der er et multiplum af $ab$. Hvis $p$ og $q$ er 100 cifrede tal, vil andelen af meddelelser, der ikke har en sådan periode være

$$\frac{6a + 6b - 3}{(2a+1)(2b+1)} \approx 10^{-100}$$

Vi viste tidligere at

$$e^u \equiv 1 \bmod \mathrm{ORD}(m, n)$$

Derfor ved nu, at det for stort set alle meddelelser gælder :

$$e^u \equiv 1 \bmod ab \tag{9.10}$$

Vi kan nu sige noget om $u$'s størrelse. Ligning 9.10 viser at $u$ må være et multiplum af $e$'s orden i $ab$. Hvis vi vælger $a$ og $b$ som ekstra stærke primtal, får vi

$$a \equiv 2 \cdot a' + 1 \quad b \equiv 2 \cdot b' + 1$$

hvilket betyder,at $e$'s orden i $ab$, næsten altid er et multiplum af $a'b'$, to primtal af størrelsesorden $10^{99}$. Derfor vil $u$ også mindst være af denne størrelse. Vi kan altså konkludere at gentagen indkodning er chanceløs for stort set alle meddelelser.

Rivest har i [45] argumenteret for at det ovenstående ikke blot gælder for ekstra stærke primtal, men også for almindeligt stærke primtal. Dette gjorde han som et svar på [53], som foreslog gentagen indkodning som en metode til at bryde RSA. Diskussionen er iøvrigt fortsat i [24] og [46].

[2] har til gengæld vist, at *hvis* man finder blot et sæt $x, y$, så

$$x^y \equiv x \bmod n$$

hvor $n = p \cdot q$ og $p$ og $q$ er ekstra stærke primtal, så kan man bryde RSA-systemet og få faktoriseringen af $n$.

## 9.5 RSA-bits

Efter dels at have analyseret hvordan den hemmelige nøgle $d$ kan findes, og dels at have analyseret om enkelte meddelelser kan afkodes uden kendskab til den hemmelige nøgle, vil vi nu se på hvor svært det er at genskabe enkelte bit i en meddelelse ud fra en given ciffertekst.

På mange måder er det den ideale situation, hvis det er ligeså svært at genskabe et enkelt bit i meddelelsen, som at genskabe hele meddelelsen. Det ville betyde at RSA ikke "lækker" nogen information om klarteksten, og gøre systemet anvendeligt til andre ting, f.eks. plat og krone over telefonen. Der måtte man ikke kunne gætte, om et klarkteksten var lige eller ulige, udfra ciffertekstens.

Man har endnu ikke kunnet vise at ovenstående gælder, men har bevist følgende (fra [33]) :

| Ciffertekst | Klartekst |
|---|---|
| | |

**Tabel 9.2:** Dekryptering ved tabelopslag

**Sætning 9.5** *For ethvert RSA-system er følgende udsagn ensbetydende :*

1. *Der findes en effektiv algoritme til at afkode en meddelelse m for alle m.*
2. *Der findes en effektiv algoritme til at beregne det mindst betydende bit i meddelelsen.*

*Der gælder yderligere at hvis 1 er opfyldt, så findes der en effektiv algoritme til at beregne den mest betydende bit i meddelelsen samt, en effektiv algoritme til at beregne de s mindst betydende bit i meddelelsen.*

## 9.6 Sikkerhed ved anvendelse af RSA

Ved anvendelsen af RSA, findes der nogle faldgruber, som kan opstå under visse omstændigheder. De følgende tre afsnit tager højde for disse.

### 9.6.1 Udtømning af meddelelsesrummet

Hvis RSA anvendes i et system, hvor der er et meget begrænset antal *mulige* meddelelser, er det muligt at kompromittere sikkerheden ved at indkode samtlige meddelelser og opstille dem i en tabel sammen med de korresponderende klartekster (se tabel 9.2). Enhver ciffertekst kan herefter afkodes ved et simpelt tabelopslag.

### 9.6.2 Systemer med fast indkodningsnøgle

Som det er beskrevet i afsnittet om anvendelsen af RSA i Dankort netværket, er det af beregningsmæssige grunde ønskeligt at vælge indkodningsnøglen $e$ så lille som muligt.

Vælger man desuden $e$ lille og tilmed *ens* for alle brugere i systemet, kan sikkerheden kompromitteres af enhver udenforstående kryptoanalytiker, såfremt den samme meddelelse indkodes til $e$ forskellige brugere.

Lad $e = 3$. De tre ciffertekster er da givet ved kongruenserne

$$\begin{aligned} c_1 &\equiv m^3 \bmod n_1 \\ c_2 &\equiv m^3 \bmod n_2 \\ c_3 &\equiv m^3 \bmod n_3 \end{aligned}$$

hvor $m^3$ er den ubekendte.

Hvis $\mathrm{SFD}(n_1, n_2, n_3) = 1$ har systemet af kongruenser ifølge den kinesiske restsætning netop 1 fælles løsning modulo $n_1 n_2 n_3$: Denne løsning kan kun være $m^3$.

Da $m^3 < n_1 n_2 n_3$ kan $m$ beregnes ved at uddrage den $e$'te (her den $3^{\text{die}}$) rod af denne løsning.

Er derimod $\mathrm{SFD}(n_1, n_2, n_3) > 1$ kan systemet brydes gennem kendskab til faktorerne i $n_1, n_2, n_3$.

### 9.6.3 Systemer med fast $n$

Endelig kan man i god tro forestille sig et RSA-system, hvor alle brugere benytter samme $n$. Dette har den fordel, at det "tunge" nøgleoprettelsesarbejde lettes, idet der ialt kun skal genereres 2 primtal $p$ og $q$ til brug i alle nøgler. Desværre medfører det at sikkerheden kan brydes af både udenforstående og i endnu højere grad af brugere i systemet.

### Udenforstående angreb på system med fast $n$

En meddelelse $m$ kan dekrypteres af en udenforstående kryptoanalytiker, hvis den indkodes til to forskellige personer

$$\begin{aligned} c_1 &\equiv m^{e_1} \bmod n \\ c_2 &\equiv m^{e_2} \bmod n \end{aligned}$$

og hvis

$$\mathrm{SFD}(c_1, n) = 1, \ \wedge \ \mathrm{SFD}(e_1, e_2) = 1$$

Når dette er opfyldt har $c_1$ en invers $c_1^{-1}$ i $(Z_n)$ og der findes $R, S$ så

$$R \cdot e_1 + S \cdot e_2 = 1$$

Da både $e_1$ og $e_2$ er positive må enten $S < 0$ eller $R < 0$.

Antag at $R < 0$. Så er $R = -\,|R|$ og følgende kan beregnes:

$$\begin{aligned} (c_1^{-1})^{|R|} \cdot (c_2)^S &\equiv (m^{e_1})^{|R|} \cdot (m^{e_2})^S \\ &\equiv m^{(e_1 \cdot R + e_2 \cdot S)} \\ &\equiv m \bmod n \end{aligned}$$

Hvis det er $S$, der er negativ gælder samme argument.

### Brugeres angreb på system med fast $n$

En bruger i systemet, der har sit eget sæt nøgler, kan beregne enhver anden brugers hemmelige nøgle.

Lad brugerens eget nøglesæt være $(e_0, d_0, n)$. Angrebet består da i at beregne et $d'$ så

$$e_i \cdot d' \equiv 1 \bmod \phi(n),$$

hvor $e_i$ er den $i$'te brugers offentlige nøgle.

Antag følgende:

Der eksisterer et $x \in \mathbb{Z}$ så

1. $\mathrm{SFD}(e_1, x) = 1$
2. $x = k \cdot \phi(n)$

Da gælder det at der findes $R, S$ så

$$R \cdot x + S \cdot e_1 = 1$$

Dette medfører ifølge punkt 2 at

$$\begin{aligned} R \cdot k \cdot \phi(n) + S \cdot e_1 &= 1 \\ S \cdot e_1 &\equiv 1 \bmod \phi(n) \end{aligned}$$

Afkodningsnøgle $d'$ er derfor lig $S$.

Et $x$ som opfylder antagelserne kan findes som følger:

1. $F = \mathrm{SFD}(e_1, e_0 d_0 - 1)$
2. $x = (e_0 d_0 - 1)/F$

Dette ses af nedenstående argument

$x$ er indbyrdes primisk med $e_1$, hvilket følger direkte af punkt et og to.

$$\begin{array}{ll} F|e_1 \ \wedge & \\ \mathrm{SFD}(e_1, \phi(n)) = 1 & \Rightarrow \\ \mathrm{SFD}(F, \phi(n)) = 1 & \Rightarrow \\ x \cdot F = e_0 d_0 - 1 = k \cdot \phi(n) & \Rightarrow \\ \phi(n)|x \cdot F & \end{array}$$

Da $\mathrm{SFD}(F, \phi(n)) = 1$ må $\phi(n)|x$.

## 9.7 Sikkerhed ved RSA-signaturer

Digitale underskrifter, beskrevet i kapitel 7 er ikke altid sikre. I dette afsnit vil vi dels se på, hvordan man kan snyde med disse signaturer, og dels på en foreslået metode til at beskytte sig mod disse angreb på en signaturer.

### 9.7.1 Angreb på signaturer

En kryptoanalytiker som ønsker at signere en meddelelse fra A[1], konstruerer 3 meddelelser $m_1, m_2$ og $m_3$, som opfylder, at

$$m_3 \equiv m_1 \cdot m_2 \bmod n_A$$

Hvis hun får lokket $A$ til at signere $m_1$ og $m_2$ som $c_{s1}$ og $c_{s2}$, kan hun beregne produktet af signaturerne og derved få en falsk underskrift på $m_3$. Signaturen på $m_3$ beregnes således :

$$\begin{aligned} c_{s3} &= (m_1 \cdot m_2)^{d_A} \text{ MOD } n_A \\ &= (m_1^{d_A} \text{ MOD } n_A) \cdot (m_2^{d_A} \text{ MOD } n_A) \text{ MOD } n_A \\ &= c_{s1} \cdot c_{s2} \text{ MOD } n_A \end{aligned}$$

En tilsvarende metode er at benytte $m^{-1}$ eller $-m$ under den antagelse, at den til $m$ korresponderende underskrift er kendt. I disse tilfælde vil $c_s$ kunne beregnes som følger :

$$\begin{aligned} m^{-1} &: c_s^{-1} \equiv (m^{d_A})^{-1} \bmod n_A \text{ og} \\ -m &: -c_s \equiv -(m^{d_A}) \bmod n_A \end{aligned}$$

En anden mulighed er at vælge et heltal $c_s$, hvor $0 < c_s < n_A$ og derfra beregne

$$m_c \equiv (c_s)^{e_A} \bmod n_A$$

Gør en kryptoanalytiker dette, kan hun erklære, at $c_s$ er $A$'s underskrift på $m_c$. Da eksponentiering modulo $n$ opfører sig som en envejsfunktion, når $\phi(n)$ ikke er kendt, kan dette angreb kun bruges til at finde underskrifter på tilfældige (uforudsigelige) meddelelser, med mindre man er så heldig at finde et $m_c$, der er en meningsfyldt meddelelse. Hvis RSA-systemet f.eks. bruges til at overføre beløb mellem banker, kan næsten alle meddelelser være gyldige beløb, hvorfor man er nødt til at sikre sig mod, at sådanne uforudsigelige meddelelser har en chance for at kunne accepteres. Hertil kan man benytte redundans, således at kun meddelelser, der opfylder bestemte krav er gyldige. F.eks alle meddelelser der ender med 200 nuller.

***Eksempel 9.2. Redundans som signatur***

[1]Dette er et eksempel på anvendelse af et valgt ciffertekst angreb.

*Hvis redundansen består i, at 100 bits er nul, vil en tilfældigt valgt underskrift kun have en chance på $2^{-100}$ for at være en gyldig meddelelse. Med andre ord vil det sige, at man skal prøve i snit $2^{99}$ forskellige meddelelser for at ramme en, der er meningsfyldt. Dette gør det næsten umuligt at gætte sig frem til en sådan meddelelse.*

### 9.7.2 Signering med højre-tilføjet redundans

Denne metode, taget fra [28], bygger på, at den aktuelle meddelelse $m$ multipliceres med en kendt konstant $w$. Det betyder at alle gyldige meddelelser er på formen $m_s = w \cdot m$. De vil derfor opfylde

$$m_s \equiv 0 \bmod w$$

Sættes $w$ til en potens af to, vil der være nuller på de $\log_2 w$ mindst betydende pladser (til højre).

#### *Eksempel 9.3. Højre-tilføjet redundans*

*Et eksempel på et signatur-system med højre-tilføjet redundans, er det følgende :*

$$\begin{aligned} n = 7 \cdot 13 &= 91 \\ \textit{De aktuelle meddelelser } m &\in [0, 1, .., 15] \\ \textit{Konstant } w &= 6 \\ \textit{De gyldige meddelelser } m_s &\in [0, 6, 12, ..., 90] \\ \textit{De mulige meddelelser } m_m &= [0, \ldots, 91] \end{aligned}$$

#### Angreb mod højre-tilføjet redundans

Hvis RSA-systemet benyttes i forbindelse med denne redundans metode, kan et angreb på systemet forløbe som følger.

1. Vælg en aktuel meddelelse $m_1$, hvor $m_1 < \frac{n_A}{w}$, på hvilken man ønsker A's underskrift og beregn $m_{s1} = S_A(m_1 \cdot w)$

2. Beregn så $x \equiv (m_1 \cdot w)^{-1} \bmod n_A$. Hvis $w \leq n^{\frac{1}{2}}$, kan man ifølge [28], finde et $c$, $0 \leq c < \frac{n_A}{w}$ således at

$$cx \text{ MOD } n_A \leq \frac{n_A}{w} \quad \text{eller}$$

$$-cx \text{ MOD } n_A \leq \frac{n_A}{w}$$

Ud fra dette kan man finde to meddelelser $m_2, m_3$ således at

$$m_2 \equiv m_3 x \bmod n_A \text{ eller}$$

$$m_2 \equiv -m_3 \bmod n_A$$

og hvor $0 < m_2 < \frac{n_A}{w}$ og $0 < m_3 < \frac{n_A}{w}$.

3. Problemet kommer nu, hvor man skal opnå at få A's underskrift på meddelelserne $m_2$ og $m_3$. Hvis dette lykkes, kan man ved brug af angrebet beskrevet tidligere signere den tredje meddelelse som A ikke kender noget til, på følgende måde.

Antag at $m_2 = m_3 x$.

$$\begin{aligned} m_{s1} = S_A(m_1 \cdot w) &= S_A((x^{-1}(m_3 \cdot w)(m_3 \cdot w)^{-1}) \text{ MOD } n_A) \\ &= (S_A(m_3 \cdot w) S_A(m_3 w \cdot x)^{-1}) \text{ MOD } n_A \\ &= m_{s3} m_{s2} \text{ MOD } n_A \end{aligned}$$

Dette angreb kan kun lykkes, hvis der for meddelelserne $m_1$ og $m_2$ gælder

$$m_1 \neq m_2 \neq m_3$$

Hvis vi i vores søgen efter $m_2$ finder den lig en af meddelelserne $m_1$ eller $m_3$, kan vi prøve en anden værdi af $c$.

### Et andet angreb på Højre-tilføjet redundans

Hvis den aktuelle meddelelse $m$ for hvilken vi ønsker en falsk underskrift, kan skrives som produktet af to tal $a_1$ og $a_2$, vil $m_s$ være

$$m_s = mw = a_1 a_2 w < n$$

Vi vælger nu to tal $b_1$ og $b_2$, så der gælder

$$b_1 < a_2 \text{ og } b_2 < a_1$$

og således at

$$m_{s1} = a_1 \cdot b_1 \cdot w < n$$
$$m_{s2} = a_2 \cdot b_2 \cdot w < n$$
$$m_{s3} = b_1 \cdot b_2 \cdot w < n$$

Så vil

$$\begin{aligned} m_s &= \frac{m_{s1} \cdot m_{s2}}{m_{s3}} \\ &= \frac{a_1 b_1 w \cdot a_2 b_2 w}{b_1 b_2 w} \\ &= a_1 a_2 w \end{aligned}$$

Hvis man kan lokke A til at signere $m_{s1}$, $m_{s2}$ og $m_{s3}$, kan man konstruere en falsk underskrift på den valgte meddelelse $m$.

Denne angrebsmetode kaldes MIDO - (Multiplying-In-Dividing-Out). Forskellen på dette angreb og det ovenstående er, at dette virker for alle redundansstørrelser. På den anden side vil MIDO-angrebet ikke virke for alle meddelelser, da det kan være praktisk umuligt, at faktorisere $m$. Man kan selvfølgelig manipulere med udvalgte faktorer og derved konstruere en passende aktuel meddelelse $m$, f.eks. ved at indskyde et passende antal blanktegn.

Tilsvarende angreb kan gennemføres, hvis $m$ er tilføjet redundans til venstre (på de mest betydende bits).

## 9.8 Afrunding

Vi har i dette og i det foregående kapitel beskrevet en række vigtige resultater angående RSA-systemets sikkerhed, og opstillet en række krav til de involverede parametre. På den baggrund mener vi at RSA-systemet, med parametrene valgt som beskrevet, og med den nuværende talteoretiske viden, må siges at være et sikkert kryptosystem.

Herunder er de nødvendige krav til et sikkert RSA-kryptosystem opstillet:

**$p$ og $q$**

1. Stærke primtal

2. Mindst af størrelsesorden 50 - 100 cifre
3. $p \neq q$
4. $p$ og $q$ af samme størrelsesorden
5. afstanden mellem $p, q$ og $\sqrt{n}$ skal være stor.

**$e$ og $d$**

1. $e \cdot d \equiv 1 \bmod \phi(n)$
2. $d > \max(p, q)$
3. Alle $e$ forskellige.
4. $e \neq d$
5. $\mathrm{SFD}(e-1, p-1) = 2$ og $\mathrm{SFD}(e-1, q-1) = 2$

**Meddelelse $m$**

1. Ikke for få mulige meddelelser.

**$n$**

1. Forskellige for alle brugere.

# Del III

# Implementering

# Kapitel 10

# Design af et RSA-kryptosystem

Dette og det næste kapitel handler om hvorledes RSA-kryptosystemet kan anvendes i praksis. Dette belyses dels gennem design og implementering af de grundlæggende RSA-algoritmer, dels gennem diskussion af udformningen af et RSA-kryptosystem og dels gennem et eksempel i praksis.

I den konkrete implementering har vi valgt at lægge vægt på et pænt og struktureret design, der på en overskuelig måde kan illustrere de væsentligste aspekter af et RSA-system. Således har vi optimeret de enkelte dele i rimelig grad, men ikke i et sådant omfang, at det er gået udover programmets værdi som model og eksempel. Grunden til dette er først og fremmest at et RSA-system stiller så store krav til hurtig multipræcisionsaritmetik, at det i praksis kun kan implementeres med speciel hardware. Således er et hurtigt system udenfor en almindelig software-implementerings rækkevidde. Med dette in mente har vi valgt at benytte programmeringssproget SIMULA, der blandt andet fordi det er objektorienteret og generelt imødekommer vores behov.

## 10.1 Overordnet design

Designet har som udgangspunkt, at RSA-kryptosystemet henvender sig til to forskellige brugertyper :

1. En *programmør-bruger*, der ønsker at benytte RSA-kryptosystemet i et eller andet stykke programmel. Det kan f.eks. være i forbindelse med et database-programmel, et tekstbehandlingssystem, programmel til datakommunikation eller lignende.

2. En *slutbruger*, som for eksempel ønsker at sende og modtage kryptograferede meddelelser, at lave elektronisk underskrift m.m.

Et fælles træk ved begge brugertyper er, at de, når de arbejder med RSA-kryptering, ikke skal belemres med de tekniske detaljer i ind- og afkodning samt nøglegenerering, blandt andet regning med store tal og primtalstests.

Programmørbrugeren skal have adgang til et sæt af procedurer, som hun kan kalde fra sit program. Hun ønsker mulighed for at ændre på systemets funktioner (for eksempel nøglestørrelse) i det omfang, hun har behov for det. Slutbrugeren derimod, ønsker kun adgang til et færdigt program, eller en funktion i et færdigt program, der foretager ind- og afkodning.

En forudsætning for at kunne ind- og afkode, er aritmetik med heltal af størrelsesordenen 200 decimale cifre. Da et normalt programmeringssprog kun tillader præcis beregning med heltal med 5-10 decimale cifre, er det nødvendigt at udstyre systemet med rutiner, som kan foretage de nødvendige regneoperationer på sådanne store tal. Denne facilitet kalder vi et multipræcisionsværktøj.

Disse overvejelser har ledt os frem til at dele implementeringen i tre dele :

1. Et multipræcisionsværktøj til at repræsentere og regne med store heltal.

2. Et RSA-værktøj til programmør-brugeren. Dette værktøj skal kunne håndtere de grundlæggende algoritmer : ind- og afkodning, samt nøglegenerering.

3. Et færdigt program til slutbrugeren. Dette program skal kunne ind- eller afkode en meddelelse ved kald af en enkelt kommando.

I dette og det næste kapitel beskriver vi design, implementering og test af multipræcisions- og RSA-værktøjet (punkt 1 og 2). I kapitlet derefter diskuterer vi de væsentlige punkter i designet af et RSA-slutbrugerprogram (punkt 3).

## 10.2 RSA-værktøjets brugergrænseflade

Vi vil nu se på, hvordan RSA-værktøjet skal designes. Gennem en analyse af brug og brugere, vil vi argumentere for værktøjets brugergrænseflade.

### 10.2.1 Brug og brugere

Der er grundlæggende to forskellige anvendelser af RSA-programmør værktøjet.

- Den hvor RSA-værktøjet bruges i et selvstændigt kryptosystem, hvor al ind- og afkodning foregår af samme program. Her er det ikke nødvendigt at tage højde for forskellige andre standarder.
- Den hvor RSA-værktøjet skal bruges sammen med andre RSA-systemer, f.eks. i et netværk, hvor meddelelser kan være indkodet efter forskellige konventioner. For eksempel kan meddelelser være indkodet med forskellige blokstørrelser, med eller uden cifferblokkobling, signerede eller ikke signerede. Altsammen noget der kræver et system, der kan følge de forskellige konventioner.

I den første anvendelse kan brugeren principielt set overlade hele arbejdet til systemet. Det drejer sig om styring af sikkerhedsniveauet, konvertering mellem tekst og tal m.m. I den anden anvendelse findes der mange konventioner til at behandle meddelelser, der skal ind- og afkodes. Brugeren af værktøjet skal derfor have kontrol over konfigurationen af de parametre, der styrer opsætningen af et RSA-system. Disse to typer anvendelser leder frem til følgende opdeling af programmørbrugerne :

- *Den "dumme" programmør-bruger.*
  Han er kun interesseret i at meddelelserne indkodes og afkodes – ikke i hvordan det sker. Principperne bag systemet er ham ligegyldige.
- *Den "avancerede" programmør-bruger.*
  Hun har brug for, eller ønsker at have kontrol over kodningsprocessen. Hun kender RSA-systemet og principperne bag det og ved også en del om kryptografi.

Da de brugerne af vores system er placeret i eller mellem disse to kategorier, skal vores system, for at være så anvendeligt som muligt, tilgodese begge brugertyper. Dermed tilgodeser vi også begge typer anvendelser.

### 10.2.2 Designkrav

#### Krav – Udformning

Inden vi ser på hvilke funktioner, der skal ligge i RSA-værktøjet, vil vi opstille nogle generelle krav til systemets udformning. De er sammenfattet i de følgende punkter :

1. *Enkelhed.*
   Det er vigtigt at systemet er let at gå til, og at det ikke kræver et dybt kendskab til mekanismerne bag RSA-kryptosystemet, specielt for den "dumme" brugers skyld. Samtidig bør grænsefladen kun indeholde forholdsvis få og let anvendelige funktioner.

2. *Fleksibilitet.*
   Grænsefladen bør ikke designes med henblik på nogen bestemt type applikation. Den bør give mulighed for at systemet på en naturlig måde kan indgå i forskellige typer applikationer. Samtidig bør systemet kunne tilpasses forskellige udgaver af RSA-kryptosystemet.

3. *Gennemskuelighed.*
   Grænsefladen bør ikke give anledning til en speciel, knudret programmeringsstil hos brugeren. Den bør i stedet afspejle hvad der sker i programmet, således at det bliver så let gennemskueligt som muligt.

4. *Driftsikkerhed.*
   Grænsefladen bør være designet på en sådan måde, at brugeren har så få chancer som muligt for at lave fejl i anvendelsen. F.eks. i form af ugyldige parametre. De steder, hvor der kan begås fejl, bør brugerens program have besked om dem, så det kan reagere på en fornuftig måde på fejlen. Det er ikke RSA-værktøjets opgave at udskrive fejlmeddelelser. Vi ved jo ikke hvordan der kommunikeres med brugerne i programmet, om der bruges grafik, vinduer eller almindelig tekst – eller om det overhovedet foregår on-line.

### Krav – Funktioner

Uanset anvendelse er det vigtigste krav til ethvert RSA-værktøj naturligvis, at det kan foretage de tre grundlæggende operationer :

1 Nøglegenerering.

2 Indkodning.

3 Afkodning.

Da sikring af autenticitet via signering og verifikation, normalt foregår med den samme algoritme som ind- og afkodning, vil det være enkelt og derfor naturligt også at give brugerne denne mulighed :

4 Signering.

5 Verifikation.

Da det originale signeringsskema er sårbart overfor visse typer angreb som beskrevet i kapitel 9, og da signering og verifikation ikke er en fast defineret del af RSA-kryptosystemet, bør der dels være mulighed for at vælge det fra, og dels være mulighed for, at den avancerede bruger *selv* kan angive en signerings/verifikations algoritme.

Ud over de ovenstående funktioner er der også nogle styringsparametre, som vi mener er vigtige at have med i et RSA-kryptosystem.

### Sikkerhedsniveauet

Det bestemmes primært gennem "modulussen" $n$'s størrelse. $n$ skal derfor konstrueres, så den bliver større end et valgt sikkerhedsniveau. Vi kan enten vælge at lade sikkerhedsniveauet have en fast værdi i systemet, eller lade det være brugerstyret. Her er det sidste at foretrække, ikke mindst fordi faktoriseringsalgoritmerne, som tidligere beskrevet, bliver hurtigere og hurtigere år for år. Herved mindskes sikkerheden af en bestemt størrelse $n$, og det kan være nødvendigt at skifte til en større værdi af $n$.

Den "dumme" og den "avancerede" brugers krav kan her, som andre steder i systemet, forenes ved samtidigt at sørge for at sikkerhedsparameteren fra starten har en (rimelig) standardværdi, *og* at den kan ændres, hvis det synes nødvendigt.

### Blokstørrelse

RSA-kryptosystemet benytter blok-indkodning (se kapitel 6), uden at der er andre krav til blokkenes størrelse end at deres talrepræsentation skal ligge i intervallet $[0, n-1]$. Dette gør at forskellige systemer kan bruge forskellige blokstørrelser. For at sikre at vores system kan bruges i forbindelse med andre systemer, bør der være mulighed for, at brugeren selv styrer størrelsen af blokkene.

Hvis værktøjet bruges alene, og ikke skal kommunikere med andre systemer, kan brugeren være ligeglad med hvordan teksten opsplittes i blokke. Derfor bør der også her være mulighed for at brugeren selv angiver en algoritme til opsplitningen, samtidig med at der defineret en standard-opsplitning, som anvendes, hvis ikke andet er angivet.

### CifferBlokKobling

I kapitel 6 beskrev vi, hvordan et system, der benytter blok-indkodning, kunne afsløre den overordnede struktur i en meddelelse, og vi beskrev også modtrækket mod dette : *CifferBlokKobling - CBK*. Da RSA-kryptosystemet bruger blok-indkodning, vil det være naturligt at tilbyde muligheden for cifferblokkobling, som en styringsparameter. Standardmetoden til at lave CBK er beskrevet i kapitel 6, men man kan sagtens forestille sig andre udgaver. Derfor bør der også her være mulighed for at vælge CBK ja/nej, og yderligere bør brugeren selv kunne angive en CBK-algoritme.

Alt i alt stiller vi følgende krav til hvad systemet skal kunne :

| | | |
|---|---|---|
| 1 | Nøglegenerering | |
| 2 | Indkodning | |
| 3 | Afkodning | |
| 4 | Signering | Valgfrit med valgfri algoritme |
| 5 | Verifikation | Valgfrit med valgfri algoritme |
| | | |
| A | Sikkerhedsniveau | Valgfrit |
| B | Blokstørrelse | Valgfrit |
| C | CifferBlokKobling | Valgfrit med valgfri algoritme |

For alle valgfri parametre og algoritmer bør der være defineret standardværdier og standard-algoritmer.

### 10.2.3 Designovervejelser

Gennem diskussionen i det forrige afsnit har vi beskrevet hvilke funktioner, RSA-værktøjet skal kunne udføre. Disse krav vil danne udgangspunktet i det følgende.

#### Struktur

Vi har overvejet to modeller for værktøjets funktionsmåde.

1. En utraditionel, hvor SIMULA's input/output funktioner udbygges til at give mulighed for RSA-kryptografi.
2. En traditionel, hvor grænsefladen består af en række procedurer til at udføre funktionerne.

Den sidste model er den mest umiddelbare, mens den første er den mest elegante. SIMULA's opbygning muliggør nemlig, at den førstnævnte kan gennemføres i sproget, næsten transparent for brugeren ved at definere en speciel FIL-type (FILE CLASS RSAIO), der bruger indkoder alt hvad der skrives til fil og afkoder alt hvad der læses. Et program kunne så se ud som på figur 10.1. Når man skriver til en fil af RSAIO-typen indkodes der ved udskrift og afkodes ved indlæsning med SIMULA's normale input/output rutiner. Denne opbygning ville gøre det meget simpelt at ændre programmet fra ikke at bruge indkodning, til at gøre det. Man skal blot omdirigere ud og indlæsningen til en RSAIO fil.

Umiddelbart virker den første model smartest, specielt for den 'dumme' bruger. Den har imidlertid nogle uheldige sider.

Først og fremmest binder man kryptograferingen til en bestemt filtype. Man kan således ikke uden videre bruge systemet på andre typer - f.eks i en database, eller foretage kryptering til en udgående linie, hvis det måtte ønskes. Det andet kritikpunkt er, at modellen ikke gør brugerprogrammerne specielt gennemskuelige. Det kan blive svært at se hvornår der kryptograferes, og hvornår der ikke gør det. Endvidere

```
external class rsa;

begin
  ref(RSAIO) kodefil;
  ...
  kodefil :- new RSAIO(filnavn);
  kodefil.Open(n, e, d);
  ...
  outtext ("blablabla"); ! Udskrift i klartekst ;
  inspect kodefil do
  begin
    ...
    outtext ("blablabla") ;  ! Her indkodes der ;
    ...
    inimage ;                ! Her afkodes der ;
    t :- intext(80);         !
    ...
  end;
  ...
end ** program ** ;
```

**Figur 10.1:** 1. designforslag til RSA-værktøjet

```
external class rsa;

begin
  ...
  outtext ("blablabla") ;       ! Udskrift i klartekst ;
  ...
  indkod ("blablabla", n, e) ; ! Indkodning ;
  ...
  outtext ("blablabla") ;       ! Udskrift i klartekst ;
  ...
  afkod ("blablabla", n, d) ;  ! Afkodning ;
  ...
  inimage ;                     ! Indlæsning i klar- ;
  t :- inint ;                  !              tekst ;
  ...
end ** program ** ;
```

**Figur 10.2:** 2. designforslag til RSA-værktøjet

er modellen meget SIMULA-specifik. Det er kun meget få programmeringssprog, hvor input/output funktionerne lader sig omdefinere. Det gør at programmet kun vanskeligt lader sig omskrive til et andet sprog, f.eks. med henblik på optimering eller tilpasning til specifikke systemer, som måske skal skrives i et andet sprog.

Der er altså anker mod denne løsning, på tre af de punkter vi har beskrevet som væsentlige designkrav :

- Fleksibilitet (sprog, anvendelse).
- Enkelhed (overskuelighed).
- Gennemskuelighed.

Den anden mere "kedelige" løsning lider ikke af disse problemer. Da de forskellige funktioner repræsenteres af selvstændige procedurer, er det lettere at gennemskue, hvad der foregår. Videre kan man lave overbygninger der tilpasser funktionerne til den enkelte applikation, så funktionsmåden bliver endnu klarere. Antag f.eks. at der både skal indkodes og signeres. Brugeren kan så programmere en ny procedure send der kalder de to rutiner i vores system. Endelig lader disse procedurekald sig også let omskrive til andre programmeringssprog, så systemet kan optimeres/tilpasses.

Et program af denne type kan se ud som på figur 10.2.

Vi har derfor valgt denne sidste løsning. Bemærk iøvrigt at den også giver mulighed for, at den første model kan implementeres som en overbygning.

### Dialog

Vi har nu lagt os fast på dels hvilke funktioner RSA-værktøjet skal indeholde og dels på brugergrænsefladens overordnede struktur. Vi vil nu se på hvordan grænsefladen mere præcist skal udformes.

Først har vi overvejet, hvordan de forskellige inddata i form af nøgler, klar- og ciffertekster samt styringsparametre skal overføres. Valget står mellem at gøre dem til parametre til procedurerne eller lade dem være globale variable. For inddata der ændrer sig fra procedurekald

til procedurekald, nøgler samt ciffer- og klartekster, er det mest naturligt at overføre dem som parametre. Overvejelserne kommer ind ved styringsparametrene : sikkerhedsniveau, CBK m.m.

Et argument for at lade dem overføre som parametre er, at man er fri for at skulle sætte en lang række variable, inden der kan kodes, og at det gør grænsefladen mere enkel : Alle inddata overføres på den samme måde. Et argument for at opbevare dem i globale variable er, at det giver mulighed for at sætte dem til standard-værdier, som brugeren kun behøver at ændre, hvis hun ønsker noget andet. Endvidere gør det procedurekaldene enklere, eftersom der skal overføres færre parametre. Denne sidste mulighed gør således livet mere enkelt for den 'dumme' bruger, da han overhovedet ikke behøver at kende disse styringsparametre, samtidig med at det ikke forhindrer den avancerede bruger i noget. Derfor har vi valgt denne mulighed.

Et andet punkt vi har overvejet, er hvordan de forskellige parametre skal repræsenteres, som tekster, tal eller noget helt tredje. Klar- og ciffertekster er, som navnene siger tekster, og repræsenteres derfor naturligt som tekster i SIMULA. Hvis klar- og ciffertekserne er store, og f.eks. opbevares i en fil, kan dette skabe problemer, da det vil være meget pladskrævende at skulle opbevare en hel fil i en tekstvariabel. Man kunne derfor forestille sig at der skulle være mulighed for at overføre navnet på ind- og uddatafilen, og lade RSA-værktøjet om resten. Denne facilitet har vi ikke implementeret, men det kan forholdsvis let gøres.

Nøglerne $n$, $e$ og $d$ er tal. De kan imidlertid ikke direkte overføres som tal, da de er meget store, 100-200 cifre lange. Senere i dette kapitel beskriver vi det værktøj, vi har lavet til at regne med disse store tal samt en metode til at repræsentere dem. Denne interne repræsentation har vi imidlertid valgt *ikke* at benytte til opbevaring og overførsel af nøglerne. Ikke på grund af manglende tillid til repræsentationen, men fordi vi ønskede en enklere repræsentation, som passede mere med hvordan nøglerne naturligt gemmes mellem program-kørslerne – som tekster. En anden grund til at vi ikke har valgt den omtalte repræsentation er, at vi har ønsket en repræsentation der er *uafhængig* af multipræcisionsværktøjet, så dette kan ændres og optimeres, uden at det får indflydelse på RSA-værktøjets grænseflade. Endvidere har tekst-repræsentationen den fordel, at den ikke kræver at brugeren konverterer nøglerne inden ind- og afkodningsrutinerne kaldes. Endelig er tekst-repræsentationen

mere sprog-uafhængig, så vores system kan ved brug af denne repræsentation omlægges til andre sprog, uden at grænsefladen skal ændres. En tekst, der kun indeholder (decimale) tal, kalder vi fremover en decimaltekst.

### 10.2.4 Design af de enkelte funktioner

Vi vil nu se på hvordan de enkelte funktioner skal se ud.

#### Nøgleoprettelse

Proceduren nøgleoprettelse skal generere ind- og afkodningsnøglerne $n$, $e$, $d$ og samtidig levere de to primtal $p$ og $q$. De sidstnævnte skal, for at give maksimal sikkerhed, genereres som stærke primtal (se kapitel 9).

Inddata til nøgleoprettelsen skal i hvert fald være sikkerhedsniveauet. Det kan angives på mange måder. Vi har valgt at lade brugeren angive det, som det mindste antal decimale cifre $n$ må indeholde. Sikkerhedsniveauet er en styringsparameter, og overføres derfor, som ovenfor beskrevet, via en global variabel.

Den anden parameter, som nøgleoprettelsen skal have er den, der angiver med hvilken sikkerhed et primtal skal testes (se kapitel 4). Da man må formode, at denne sikkerhedsvariabel er nogenlunde konstant over længere perioder er også den overført, som en global styringsparameter.

Endelig skal nøgle oprettelsen have en sæd til sin tilfældighedsgenerator. Denne værdi skal af sikkerhedsgrunde ændres ved hvert kald. Derfor er den ikke global.

#### Indkodning

RSA-algoritmen skal have et heltal som inddata, og returnerer også et heltal. Vi kunne derfor have valgt at lade klar- og ciffertekster overføre som store heltal, i form af decimaltekster. Det ville imidlertid kræve at brugeren selv konverterede klarteksten til decimaltekst, en unødvendig komplikation for den 'dumme' bruger. Vi har derfor valgt at lade brugeren levere en almindelig tekst, og få en almindelig tekst tilbage. RSA-værktøjet benytter så en konverteringsrutine til at regne frem og tilbage mellem almindelige tekster og decimaltekster. For at sikre at værktøjet kan bruges generelt, får brugeren mulighed for at angive sin

egen konverteringsalgoritme, hvis han ønsker det. Så sikrer vi os mod problemer, hvis man skal sende en ciffertekst til et andet RSA-system, der benytter en anden konverterings-algoritme.

RSA-algoritmen arbejder på heltal i intervallet $[0, n-1]$. Da brugeren ikke ser den decimale repræsentation af sine klar- og ciffertekster, kan han ikke selv splitte dem op i blokke af passende størrelse. Derfor skal inddata ikke blot være en klartekstblok, men hele klarteksten. En anden grund er at CBK-algoritmen arbejder på flere ciffertekstblokke ad gangen, og derfor heller ikke kan nøjes med en blok. Da forskellige RSA-systemer kan splitte blokkene op på forskellig måde, skal der også her være mulighed for at brugeren kan specificere sin egen opsplitningsalgoritme, hvis det ønskes.

Udover en klartekst, skal indkodningsproceduren også have indkodningsnøglen $(n, e)$ som inddata. De repræsenteres som tidligere beskrevet som to decimaltekster.

Endelig skal brugeren angive om der ønskes CBK eller ej. Dette er en boolsk styringsparameter, og sættes derfor gennem en global logisk variabel. Også CBK-algoritmen kan ændres, præcis som for de ovenstående funktioner.

### Afkodning

Afkodning foregår modsat indkodning. Ind- og uddataparametrene er derfor stort set de samme. Blot er der byttet om, så ciffertekst en er inddata og klarteksten er uddata. Desuden er den offentlige nøgle $(n, e)$, skiftet ud med den hemmelige nøgle $(n, d)$.

### Signering/verifikation

Signering og verifikation er to selvstændige procedurer. De kunne være sparet bort, idet de er identiske med af- og indkodning, eller de kunne have været valgt via parametre til indkodning og afkodningsrutinerne. Vi har imidlertid valgt at lade dem være selvstændige for at gøre tingene så fleksible og gennemskuelige som muligt. De har naturligvis de samme inddata og uddata som henholdsvis afkodning og indkodning.

### Brugerstyrede faciliteter

De ting som ethvert RSA-system skal opfylde er :

- Ind- og afkodningsalgoritmen.
- Kravene til $p, q, n, e, d$.
- At intervallet for en decimaltekstblok skal være $[0, n-1]$.

Disse ting kan der derfor ikke ændres på.

De ting der skal kunne ændres, er de funktioner, som kan være forskellige fra system til system. Det er de følgende :

- Sikkerhedsniveauet (og dermed størrelsen af $n$).
- CBK ja/nej.
- CBK-algoritme.
- Signering/verifikation's algoritmer.
- Konvertering mellem klartekst/ciffertekst og decimaltekster.
- Måden hvorpå en tekst opsplittes i blokke.

Specielt for CBK-algoritmen og de to sidste punkter gælder, at højniveau implementeringen tager udgangspunkt i teksternes decimale repræsentation. Derfor får man et andet resultat, end hvis der regnes binært. F.eks. består en blok af et helt antal decimale cifre, ikke af et helt antal binære cifre.

For på den ene side at lade den 'dumme' bruger slippe for at tænke på disse ting, og samtidig give den 'avancerede' bruger mulighed for at styre tingene, har vi valgt den fleksible løsning. På den ene side har vi lavet procedurer til at ordne disse ting, og lavet globale variable til at holde styringsparametrene. På den anden side har vi *virtual-specificeret* de procedurer, der udfører de ovennævnte funktioner, således at brugeren kan angive sin egen udgave. De eneste krav der stilles til brugerens procedurer er at de overholder nogle få konventioner mht. ind- og uddataparametrene. Dette kan også efterlignes i mange andre sprog, således at det ikke binder vores RSA-system til SIMULA.

### Fejlhåndtering

Vi har allerede omtalt at brugeren af RSA-værktøjet, skal kunne reagere på de fejl der måtte opstå undervejs. Eksempler på fejl er f.eks. at der overføres forkerte parametre til en procedure o.l. For at sikre at der altid bliver taget hånd om de fejl der måtte opstå, og for samtidig at give brugeren mulighed for selv at reagere på dem, har vi også her virtual specificeret fejlproceduren. Vi har lavet en enkel procedure, der udskriver en meddelelse i tilfælde af fejl og som træder i kraft, hvis ikke andet er specificeret.

For at brugeren kan reagere fornuftigt, bør han have oplysninger om hvor alvorlig fejlen er, om dens type, og om hvor den er opstået. Derfor har fejl-proceduren tre inddata-parametre :

- Alvorlighed - et tal $\in [1, 10]$
- Type - Et symbol, der angiver arten, f.eks `DIVNUL` for division med nul.
- Navnet på den rutine hvor den opstået.

## 10.2.5 Grænseflade - præsentation

Tilsammen kommer vores brugergrænsefladen af RSA-kryptosystemet til at se ud som følger :

```
Class rsa;
Begin

  Virtual : Procedure fejl,
            Procedure konverter_ind,
            Procedure konverter_ud,
            Procedure blok_opdel,
            Procedure saml_blokke,
            Procedure cbk_ind,
            Procedure cbk_ud,
            Procedure signering,
            Procedure verificering;
```

```
********** ind- og afkodningsrutiner ***********;

text procedure indkodning (klartekst, n, e);
     text klartekst; decimaltext n, e;
text procedure afkodning (ciffertekst, n, d);
     text ciffertekst; decimaltext n, d;
text procedure signering (klartekst, n, d);
     text klartekst; decimaltext n, d;
text procedure verificering (signeret_tekst, n, e);
     text signeret_tekst; decimaltext n, e;

********** brugerstyrede rutiner ************;

Procedure Fejl (alvor, type, rutine);

decimaltext Procedure konverter_ind (tekst);
            text tekst;
text Procedure konverter_ud (tekst);
            decimaltext tekst;
decimaltext Procedure blok_opdel (tekst);
            decimaltext tekst;
decimaltext Procedure saml_blokke (tekst);
            decimaltext tekst;
decimaltext Procedure cbk_ind (m_blok, c_blok);
            decimaltext m_blok, c_blok;
decimaltext Procedure cbk_ud (m_blok, c_blok);
            decimaltext m_blok, c_blok;

************* nøgle generering *************;

procedure nøglegenerator (n, e, d, p, q);
          decimaltext n, e, d, p, q;


************ globale variable **************;

integer sikkerhedsniveau,
        primtalssikkerhed;

boolean CBK;
```

```
    sikkerhedsniveau := 200;  ! decimale cifre;
    cbk              := true;

end  ** class rsa **;
```

Bemærk at `decimaltext` ikke er en datatype. I den rigtige implementation er alle decimaltekster blot af type `Text`. Her har vi blot villet gøre opmærksom på hvordan teksterne skulle se ud.

Et testprogram som benytter RSA-programmør værktøjet, kunne se således ud :

```
external class rsa;

begin
  sikkerhedsniveau := 100;
  ...
  nøgler(n, e, d);
  ...
  ciffertekst := indkodning ("blablabla", n, e) ;
  ...
  klartekst   := afkodning  ("blablabla", n, d) ;
  ...
end ** program ** ;
```

Her benyttes cifferblokkobling og brugeren har valgt et lavere en sikkerhedsniveau end standardniveauet, $n$ skal have 100 decimale cifre.

## 10.3 Design af multipræcisionsværktøjet

Vi har allerede argumenteret for at det er nødvendigt at have rutiner til regning med store heltal. Uden dem, intet RSA-kryptosystem. Vi kunne have spredt disse rutiner ud i RSA-værktøjet, men har i stedet valgt at samle disse rutiner i særskilt værktøj, multipræcisionsværktøjet. Det sker af tre grunde :

- SIMULA's opbygning gør at det er muligt at lave regneapparatet næsten transparent for brugeren ved at placere det i særskilt klasse.
- Ved at samle operationerne samler man også optimeringsproblemerne. Optimering af RSA-værktøjet består hovedsageligt i optimering af multipræsicionssværktøjet, da alle de beregningsmæssigt tunge operationer ligger her, eller hviler på operationer herfra.
- Et redskab til aritmetik med store heltal, kan have andre anvendelser end RSA. Det er derfor praktisk at kunne bruge det selvstændigt.

I det følgende vil vi argumentere for designet af multipræcisionsværktøjet, først repræsentationen af tallene, og derefter funktionerne i værktøjet.

### 10.3.1 Designkrav

Vi vil her opstille de generelle krav til multipræcisionsværktøjet :

1. *Fleksibilitet.*
   Man bør tilstræbe at værktøjet fremstår så generelt som muligt, således at det ikke kun kan bruges til opbygning af RSA-værktøjet, men let kan udvides til andre formål.
2. *Hastighed.*
   Det er meget vigtigt at værktøjet er hurtigt. I modsætning til RSA-værktøjet er det på dette niveau muligt at vinde megen tid ved at optimere de enkelte rutiner. Derfor bør der vælges hurtige algoritmer til at udføre funktionerne.
3. *Driftsikkerhed.*
   Det er også her meget vigtigt, at systemet ikke "går ned" på grund af brugerfejl.
4. *Gennemskuelighed.*
   Grænsefladen bør kunne bruges ligeså naturligt, som man bruger de almindelige enkeltpræcisionsrutiner i programmeringssproget. Dette betyder, at eksisterende programmer, der opererer på enkeltpræcisionstal, forholdsvis let skal kunne oversættes til multipræcisions programmer.

### 10.3.2 Repræsentation

Helt grundlæggende er et multipræcisionstal $mt$, som almindelige heltal, repræsenteret ved en række cifre i et positionstalsystem, med et eller andet grundtal ($b$) :

$$mt = (c_n, c_{n-1}, \ldots, c_1, c_0)_b$$

Med grundtallet $b$ bliver tallet

$$mt = c_n \cdot b^n + c_{n-1} \cdot b^{n-1} + \ldots + c_1 \cdot b + c_0$$

Der er tre oplagte måder at repræsentere et sådant tal på :

1. *Decimaltekster.*
   Multipræsicionstallene kunne repræsenteres som decimaltekster, dvs. som tekststrenge af decimale cifre. Det ville overflødiggøre konvertering ind og ud af multipræcisionsværktøjet, da tallene allerede repræsenteres som decimaltekster i RSA-værktøjet. Imidlertid er det ikke praktisk muligt at regne direkte med decimaltekster. Der skal hele tiden pilles cifre ud af tallet, og hver gang skal der konverteres mellem tegn og tal. Derfor ville denne repræsentation gøre multipræcisionsværktøjet meget langsomt.

2. *Arrays af heltal.*
   Den anden repræsentationsmulighed er som arrays af heltal. Et multipræcisionstal ville så være et enkelt-indiceret array hvor hver plads indeholdt et heltal i intervallet $[0, \mathit{grundtal} - 1]$.

3. *Liste af heltal.*
   Den tredje mulighed er at repræsentere multipræcisionstallet som en (Simset) liste af heltal. Som ovenfor indeholder hvert element i listen et ciffer i multipræcisionstallet.

Den første repræsentationsform er som nævnt for langsom. Valget står derfor mellem de to andre repræsentationsformer. De er forholdsvis ens. Forskellen mellem disse to ligger primært i hvordan cifrene hægtes sammen. Derfor skulle man også tro at de hastighedsmæssigt også var ret ens. H.B. Hansen fra RUC's datalogi-uddannelse nævner dog i en note, at Simset-repræsentationen er noget hurtigere. Årsagen er her, at arrayrepræsentationen involverer dynamisk oprettelse af arrays, hver

gang resultatet af en regneoperation er blevet større end argumenterne. Det samt den efterfølgende kopiering tager uforholdsmæssigt lang tid. Dette argument har vi taget til os, og blandt andet derfor valgt simset-repræsentationen. En anden fordel ved denne repræsentation er at den er mere gennemskuelig - forrige og næste ciffer nås ved suc og `pred`, og ikke med `ciffer-1` og `ciffer+1`, som man ville gøre i array-repræsentationen. Endvidere er det lettere at tilføje et ciffer, eller pille et ciffer ud af et tal. Dette gøres med enkle Simset-ordrer `into`, `precede` og `out`.

Ulempen ved simset-repræsentationen er at den fylder noget mere end array-repræsentationen. For hvert ciffer i multipræcisionstallet, skal der udover selve cifferet gemmes to pointere, en til forrige og en til næste ciffer, hvor man i array-repræsentationen kun skal gemme selve cifferet. Derudover kan den være sværere at genskabe i andre programmeringssprog. Ikke alle programmeringssprog har faciliteter til at lave pointere, og formentlig er det kun SIMULA der har dobbelthægtede lister som standard. Problemet er imidlertid ikke så stort. Enten er der pointere, og så kan dobbelthægtede lister laves rimelig let, eller også er der ikke pointere, og så må man skifte til array-repræsentationen. Dette skift er imidlertid ikke så svært, da de to repræsentationsformer grundlæggende er ret ens. Det betyder at grundstrukturen i algoritmerne kan overføres uændret. Forskellene vil primært ligge i adresseringen af cifrene. Dette betyder at selv om valget mellem arrays og lister er et vigtigt valg, er det et valg, det er muligt at gøre om, dersom det skulle blive nødvendigt.

Efter at vi har lagt os fast på Simset-lister, ligger repræsentationen fast i store træk. Hele multipræcisionstallet bliver et

```
Head class heltal
```

og hvert ciffer repræsenteres ved et

```
link class ciffer
```

Hovedet kommer til at indeholde grundtallet dvs den base, som tallet er skrevet i og et fortegn. Hvert ciffer kommer til at bestå af et tal i intervallet $[0, \text{grundtal} - 1]$, evt med foranstillede nuller.

Ovennævnte repræsentation benyttes også i brugergrænsefladen. Der er altså ikke, som i RSA-værktøjet, tale om at brugeren benytter *en*

repræsentation, mens vi internt benytter en anden. Grunden til at vi har gjort dette, er af hensyn til hastigheden. Hvis der hele tiden skulle konverteres mellem multipræcisionstal og decimaltal, ville multipræcisionsværktøjet blive noget langsommere, specielt hvis de samme tal skal igennem en hel række af beregninger.

### Valg af grundtal

Vi har valgt ikke at arbejde med fast grundtal, men at give brugeren mulighed for selv at vælge det. Det har vi gjort dels for at gøre systemet så generelt som muligt, og dels fordi vi selv har brug for multipræcisionstal med forskellige grundtal. F.eks. udnytter konverteringen fra tekst til tal, at et ASCII-tegn kan udtrykkes som et ciffer i et tal med grundtal 128. Grundtallet for et multipræcisionstal skal lægges fast ved oprettelsen, og er derfor en parameter til klassen heltal.

```
Head class heltal (grundtal)
```

Grundtallet kan ligge i intervallet $[0, \lfloor\sqrt{\text{maxint}}\rfloor]$, da der skal være plads til at kvadrere et enkelt ciffer i multipræsicionstallet, og derudover lægge et andet ciffer samt en mente til.

Muligheden for variabelt grundtal, gør ikke implementationen væsentlig mere kompliceret. Skal to tal med forskellige grundtal f.eks. lægges sammen, konverteres det ene først til samme base som det andet, hvorefter beregningen kan foregå uændret. Ingen af de algoritmer vi bruger, er følsomme overfor valg af grundtal.

### Konstanterne `NUL` og `EEN`

Konstanterne nul og en har vi lavet på forhånd. Dels fordi vi selv skal bruge dem en del steder, og dels fordi andre brugere let kan tænkes at få brug for dem.

Om nul er der det at bemærke at det repræsenteres som et heltal med en tom liste. Det har den fordel at et nyoprettet heltal automatisk er initialiseret til nul. Desuden giver det færre muligheder for at konstruere et ulovligt multipræcisionstal. Hvis et tomt multipræcisionstal var ulovligt skulle der checkes for det overalt. Endelig passer denne repræsentation af nul, med den måde andre multipræcisionstal repræsenteres på. Vi fjerner nemlig altid alle foranstillede nuller, blandt andet af hensyn til sammenligningsrutinerne.

### 10.3.3 Funktioner

Efter at have lagt repræsentationen fast, vil vi nu se på hvilke funktioner værktøjet skal kunne udføre, og hvordan de skal udformes.

#### Notation

Vi har valgt at lægge alle regneoperationerne ind som funktioner i heltallet. Det betyder at regneudtrykkene kan skrives næsten som normalt i programmet. F.eks. bliver

```
a := b + c + d
```

til

```
a :- b.plus(c).plus(d)
```

og ikke

```
a :- plus (b, plus (c,d) )
```

som i *ikke* objektorienterede sprog. Man kan altså benytte sædvanlig infix-notation. Dog skal man bemærke at der ikke er noget operatorhierarki som normalt. Udtrykkene evalueres altid fra venstre mod højre, med mindre der er parenteser til angive andet. Således betyder

```
a :- b.gange(c).plus(d).gange(e)
a := (bc+d)e
```

og ikke

```
a := bc+de
```

Ønsker man at udregne dette, skal det skrives

```
a:- b.gange(c).plus(d.gange(e))
```

### Funktioner

De funktioner de skal ligge i værktøjet er først og fremmest de 4 grundlæggende regneoperationer :

- Addition.
- Multiplikation.
- Subtraktion.
- Division.

Derudover skal der ligge de aritmetiske funktioner, som er påkrævet i RSA-værktøjet :

- Største fælles divisor.
- Modulus.
- Exponentier - modulus.
- Beregning af invers.

De ovenstående funktioner, og RSA-værktøjet nødvendiggør de almindelige sammenligningsoperationer :

- Lig med.
- Større end.
- Mindre end.

Videre findes der en række andre rutiner, igen enten nødvendiggjort af multipræcisionsværktøjet selv eller RSA-værktøjet :

- Lige.
- Positiv.
- Negation.
- Kopier tallet.
- Skift grundtal.

Endelig er der tre ind- og udlæsningsrutiner, der konverterer mellem decimaltekster og multipræcisionstal, og mellem integers og multital :

- Multipræcisionstal ind.
- Multipræcisionstal ud.
- Integer ind.

Endelig findes der en række andre funktioner, primært til internt brug.

At vi har lagt os fast på netop disse funktioner, skyldes dels at vi har haft brug for dem, og dels at de er så grundlæggende, at de ikke kan undværes i et generelt multipræcisionsværktøj. At vi har valgt at nøjes med disse, skyldes omvendt dels at vi ikke har haft brug for flere og dels, at yderligere funktioner let kan lægges ind, hvis blot man har algoritmen for den pågældende funktion.

Vi har gjort en del ud af de grundlæggende aritmetiske operationer plus, minus, gange og division. Vi har nemlig valgt at lave hver af disse 4 operationer i ikke mindre end 4 udgaver. Dette er sket fordi vi under udviklingen af værktøjet erkendte, at der var behov for forskellige typer. Funktionerne findes i følgende udgaver :

- En funktion, der tager to multipræcisionstal, som argumenter.
- En funktion, der tager *et* multipræcisionstal og et enkeltpræcisionstal, som argumenter.
- En funktion af hver af ovenstående, der evt ødelægger det første argument.
- En funktion af hver af ovenstående, som bevarer det første argument.

Grunden til at type 2 findes særskilt, er for det første at vi ofte har brug for den, og gerne ville undgå at skulle lave en masse konverteringer mellem almindelige integers og multipræcisionstal. For det andet det andet vil der ofte være en tidsmæssig fordel ved at regne med et enkeltpræcisionstal : Dels skal et enkeltpræcisionstal som sagt ikke konverteres mellem tekst og multipræcisionstal, og dels skal man ikke via et `head` og nogle pointere for at få fat i tallet. Endelig bliver algoritmerne enklere når man ved at der kun er et ciffer i det ene tal. Bemærk

iøvrigt at det altid er tallet "til højre" der er en integer. I de følgende er det altså $b$ der er en integer :

$$a + b,\ a \cdot b,\ a - b,\ a/b$$

Den omvendte situation, hvor $a$ er en integer, er ikke relevant : $a + b$, $a - b$, $a \cdot b$ kan udtrykkes som $b + a$, $-b + a$ og $b \cdot a$. $a/b$ kunne være relevant, men da multipræcisionstallet $b$ næsten altid er større end $a$, vil det som regel give kvotienten nul, og resten $a$.

Grunden til type 3 funktionerne er, at der som regel kan spares tid til oprettelse af heltal, når det ene argument kan ødelægges. Ofte er det ene argument ligegyldigt som f.eks. i følgende beregning :

```
a := a + b
```

Det er ikke alle regnearterne der har kunnet laves i en destruktiv udgave, det gælder f.eks. for gange, men vi har alligevel valgt at tage den med, for fuldstændighedens skyld.

Der findes altså 4 udgaver af hver rutine. F.eks. for addition :

| | |
|---|---|
| `add` | ikke destruktiv multipræcision |
| `d_add` | destruktiv multipræcision |
| `enkeltadd` | ikke destruktiv enkeltpræcision |
| `d_enkeltadd` | destruktiv enkeltpræcision |

Denne navngivningsforskrift har vi også benyttet til de andre regneoperationer.

### Fejlhåndtering

For at sikre kompatibilitet mellem multipræcisionsværktøjet og RSA-værktøjet, benytter vi i multipræcisionsværktøjet samme system til at meddele brugeren eventuelle fejl der opstår undervejs, som i RSA-værktøjet.

### 10.3.4 Præsentation af værktøjet

Vores multipræcisionsværktøj ser ud som følger :

```
Simset
Begin

  Virtual : procedure Fejl;

  Head class Heltal (grundtal); Integer grundtal;
  Begin

    ! ******* regneoperationer ************;

    ref(heltal) Procedure Add(h);  ref(heltal) h;
    ref(heltal) Procedure Sub(h);  ref(heltal) h;
    ref(heltal) Procedure Mult(h); ref(heltal) h;
    ref(heltal) Procedure Div(h, rest);  name rest;
                             ref(heltal) h, rest;

    ref(heltal) Procedure EnkeltAdd(h);  integer h;
    ref(heltal) Procedure EnkeltSub(h);  Integer h;
    ref(heltal) Procedure EnkeltMult(h); Integer h;
    ref(heltal) Procedure EnkeltDiv(h, rest); name rest;
                             ref(heltal) rest; Integer h;

    ref(heltal) Procedure D_Add(h);  ref(heltal) h;
    ref(heltal) Procedure D_Sub(h);  ref(heltal) h;
    ref(heltal) Procedure D_Mult(h); ref(heltal) h;
    ref(heltal) Procedure D_Div(h, rest);  name rest;
                             ref(heltal) h, rest;

    ref(heltal) Procedure D_EnkeltAdd(h);  integer h;
    ref(heltal) Procedure D_EnkeltSub(h);  Integer h;
    ref(heltal) Procedure D_EnkeltMult(h); Integer h;
    ref(heltal) Procedure D_EnkeltDiv(h, rest); name rest;
                             ref(heltal) rest; Integer h;

    ! ********** Andre regneoperationer ***************;
```

```
    ref(heltal) Procedure Modulus(h); ref(heltal) h;
    ref(heltal) Procedure ExpoMod(exp, h);
                          ref(heltal) exp, h;
    ref(heltal) Procedure Invers;
    ref(heltal) Procedure sfd;

   ! ************ Sammenligninger **********************;

    boolean Procedure lig_med(h); ref(heltal) h;
    boolean Procedure mindre_end(h); ref(heltal) h;
    boolean Procedure større_end(h); ref(heltal) h;

    ! ************* Hjælperutiner ********************;

    ref(heltal) Procedure Max(h); ref(heltal) h;
    boolean Procedure Lige(h); ref(heltal) h;
    boolean Procedure Positiv; Positiv := not negativ;
    ref(heltal) Procedure Kopi;
    ref(heltal) Procedure Negeret;
    ref(heltal) Procedure normaliser;
    ref(heltal) Procedure Skift_venstre (pladser);
                                      integer pladser;
    Procedure del (a,b,p); name a,b;
                          Ref(heltal) a,b; integer p;

    ********** variable i heltallet ******************;

    boolean negativ;

  End * class Heltal *;

  Link class Ciffer(tal); integer tal;

  Procedure Fejl (alvor, type, rutine);

  ref(heltal) Procedure multital_ind (t, grundtal);
                              text t; integer grundtal;

  text Procedure multital_ud (h); ref(heltal) h;
```

```
  ref(Heltal) Procedure tal_til_multital (i); integer i;

end *** multicalc ***;
```

# Kapitel 11

# Implementering

Vi vil nu se på hvordan RSA- og multipræcisionsværktøjet er implementeret. Dette omfatter en gennemgang af de væsentligste rutiner som er benyttet, en beskrivelse af vores test-strategi og en vurdering af implementeringen. Gennemgangen bliver efter *"nedefra og op"* princippet, så algoritmerne i videst muligt omfang beskrives, før der refereres til dem.

## 11.1 Multipræcisionsværktøjet

Vores 'bibel' under arbejdet med multipræcisionsværktøjet, har været [30]. Det er herfra de fleste af de mere indviklede algoritmer er hentet, og det er den, der har stået som målestok for, hvor langt vi ville gå med at optimere programmet.

Selvom nogle af funktionerne i programmet findes i to eller fire udgaver[1], har vi idet følgende kun beskrevet en. Dette skyldes, at de bagvedliggende algoritmer stort set er ens. Enkeltpræcisionsudgaven er en simplere udgave af multipræcisionsudgaven, og den ikke-destruktive udgave er blot en overbygning på den destruktive.

På samme måde nøjes vi med at beskrive algoritmerne virkende på positive tal, da de øvrige muligheder kan reduceres til denne. I kommentarerne i programmet er der placeret fortegnstabeller ved de en-

[1] Destruktiv/ikke destruktiv, multipræcisions/enkeltpræcisions udgave. Se side 202

| x | y | Operation |
|---|---|---|
| + | + | x.Add(y) |
| + | - | -(x.Add(-y)) |
| - | + | -(-x.Add(y)) |
| - | - | -x.Add(-y) |

**Tabel 11.1:** Fortegnstabel for addition

kelte procedurer, hvor det har været relevant. Et eksempel er vist i tabel 11.1.

Efter denne indledende beskrivelse, vil vi gå over til beskrivelsen af selve algoritmerne.

### 11.1.1 Skift grundtal

Da vi har baseret de forskellige konverteringer mellem tekst, decimaltekst og multipræcisionstal på at kunne bruge et vilkårligt grundtal, er denne funktion essentiel. Grundtallet kan frit vælges i intervallet $[2, \lfloor\sqrt{\texttt{Maxint}}\rfloor]$, jævnfør forrige kapitel.

Algoritmen fungerer ved at dividere tallet igennem med det nye grundtal repræsenteret i den gamle base, indtil man får nul. Resten ved hver division, giver så cifrene i det nye tal *fra højre*. For 25 i grundtal 10, skiftet til grundtal 2, får vi for eksempel :

$$\begin{aligned} 25/2 &= 12 \text{ rest } 1 \\ 12/2 &= 6 \text{ rest } 0 \\ 6/2 &= 3 \text{ rest } 0 \\ 3/2 &= 1 \text{ rest } 1 \\ 1/2 &= 0 \text{ rest } 1 \end{aligned}$$

Det skiftede tal bliver så 11001.

#### Algoritme 11.1 Skift_grundtal

1. Sålænge tallet er større end nul gør :
   (a) Divider det igennem med nyt grundtal.
   (b) Lad resten være et ciffer i den nye repræsentation.

### 11.1.2 Konvertering mellem tekst og multipræcisionstal

#### Konvertering fra tekst til multipræcisionstal

Konverteringen fra decimaltekst til multipræcisionstal sker ad to omgange. Først indlæses teksten i et multipræcisionstal, hvis grundtal er en tierpotens, og derefter konverteres dette multipræcisionstal til det ønskede grundtal. Da at indlæsningen i første omgang foregår til en tierpotens, svarer et multipræcisionsciffer til et helt antal cifre i decimalteksten. Dette gør den første halvdel af indlæsningen rimelig enkel. Den anden halvdel klares derefter med proceduren `skift_grundtal`.

#### Algoritme 11.2 Decimaltekst → Multipræcisionstal

1. Opret et multipræcisionstal med radix $= 10^a$, hvor
$$a \leq \log \sqrt{\texttt{Maxint}}$$
2. Sålænge der er flere decimale cifre i decimal teksten, gør :
   (a) Læs $a$ tegn (fra højre side).
   (b) Indsæt dem i som et ciffer forrest i multipræcisionstallet.
3. Skift tallet til det ønskede grundtal.

#### Konvertering fra multipræcisionstal til tekst

Konverteringen fra multipræcisionstal til tekst, foregår omvendt af det ovenstående. Først skiftes grundtallet til en tierpotens. Derefter udskrives det nye multipræcisionstal i en tekst.

I den sidste del, ved vi igen at hvert multipræcisionsciffer fylder et eksakt antal pladser i decimalteksten. Vi skal derfor blot skrive cifferet ind på den rette i teksten, og fylde op med nuller. Eksemplet illustrerer dette.

***Eksempel 11.1. Udskrift af et multipræcisionstal***

*Grundtal = 10000. Multipræcisionstal = 1234 0048*

*Cifrene i multipræcisionstallet er :* [1234] *og* [ 48]

*48 skrives bagest i resultatteksten, og der fyldes op med nuller :*

[ 0048]

*Derefter skrives 1234 også ud, resultatet bliver da :*

[12340048]

*Hvis vi blot havde sat tallene ind i resultatteksten, havde vi fået fjernet nullerne før 48, og resultatet var blevet forkert.*

### 11.1.3 Addition og Subtraktion

Algoritmerne bygger på den metode, man benytter, når man lægger tal sammen henholdsvis trækker dem fra hinanden, ved brug af papir og blyant, som vist i eksemplet :

***Eksempel 11.2. Addition med papir og blyant***

```
   11111                    10
  234567890          1234567890
+  23456789        -   23456789
------------       ------------
 1258025679         1211111101
```

**Algoritme 11.3 Addition**

1. Byt tallene om, så det største tal står først. Indsæt underforståede nuller i det andet, så de er lige lange.
2. Initialiser menten til 0.
3. For hvert ciffer i det første tal gør følgende :
   (a) Adder dette ciffer, og det tilsvarende ciffer i det andet tal, sammen med en eventuel mente.

(b) Hvis summen er større end grundtallet, så sæt menten til 1 og lad cifferet i resultatet være sum - grundtal, ellers sæt menten til 0 og lad cifferet i resultatet være summen.

4. Hvis menten er 1 efter addition af de forreste cifre, så indsæt et nyt ciffer med menten forrest i resultatet.

Subtraktionen foregår stort set på samme måde, her skal der blot holdes øje med et lån, i stedet for en mente.

### 11.1.4 Multiplikation

Papir-og-blyant metoden til multiplikation er illustreret i eksemplet :

***Eksempel 11.3. Multiplikation med papir og blyant***

| | | | | | $123 \cdot 46$ |
|---|---|---|---|---|---|
| $10^0 \cdot 10^0$ | : | 243 | · | 6 | 18 |
| $10^0 \cdot 10^1$ | : | 3 | · | 40 | 120 |
| $10^1 \cdot 10^0$ | : | 20 | · | 6 | 120 |
| $10^1 \cdot 10^1$ | : | 20 | · | 40 | 800 |
| $10^2 \cdot 10^0$ | : | 100 | · | 6 | 600 |
| $10^2 \cdot 10^1$ | : | 100 | · | 40 | 4000 |
| | | | | | 5658 |

Denne måde at multiplicere på, har som beskrevet i kapitel 3 kompleksiteten $\mathcal{O}(n^2)$, hvor $n$ er antallet af cifre i begge faktorer. Imidlertid kan multiplikation foretages en del hurtigere ved brug af følgende metode, hentet fra [30]. Lad

$$u = (u_{2n-1} \dots u_1 u_0) \text{ og } v = (v_{2n-1} \dots v_1 v_0)$$

være to $2n$-cifrede binære tal der skal multipliceres, hvori $u_i$ og $v_i$ er de enkelte cifre. Tallene $u$ og $v$ kan da opsplittes i to lige store dele og skrives som

$$\begin{aligned} u &= 2^n U_1 + U_0 \\ v &= 2^n V_1 + V_0 \end{aligned}$$

hvor

$$\begin{array}{lcll} U_1 & = & (u_{2n-1} \ldots u_n) & \text{den } \textit{mest betydende halvdel} \text{ af } u \text{ og} \\ U_0 & = & (u_{n-1} \ldots u_0) & \text{den } \textit{mindst betydende halvdel.} \end{array}$$

På samme måde er

$$\begin{array}{lcll} V_1 & = & (v_{2n-1} \ldots v_n) & \text{den } \textit{mest betydende halvdel} \text{ af } v \text{ og} \\ V_0 & = & (v_{n-1} \ldots v_0) & \text{den } \textit{mindst betydende halvdel.} \end{array}$$

Produktet $u \cdot v$ er ved brug af *papir og blyant* metoden da :

$$\begin{aligned} uv &= (2^n V_1 + V_0)(2^n U_1 + U_0) \\ &= 2^{2n} U_1 V_1 + 2^n (U_1 V_0 + V_1 U_0) + U_0 V_0 \end{aligned} \tag{11.1}$$

Således viser ligning 11.1 at multiplikation af to $2n$-cifrede tal kan reduceres til fire multiplikationer af $n$-cifrede tal. Dette kan forbedres ved følgende trick :

$$\begin{aligned} uv &= uv + 2^n (U_1 V_1 - U_1 V_1 + U_0 V_0 - U_0 V_0) \\ &= 2^{2n} U_1 V_1 + 2^n (U_1 V_0 + V_1 U_0) + U_0 V_0 \\ &\quad + 2^n U_1 V_1 - 2^n U_1 V_1 + 2^n U_0 V_0 - 2^n U_0 V_0 \\ &= (2^{2n} + 2^n) U_1 V_1 + (2^n + 1) U_0 V_0 \\ &\quad + 2^n (U_1 V_0 + V_1 U_0 - U_1 V_1 - U_0 V_0) \\ &= (2^{2n} + 2^n) U_1 V_1 + (2^n + 1) U_0 V_0 + \\ &\quad 2^n (U_1 - U_0)(V_0 - V_1) \end{aligned} \tag{11.2}$$

$u \cdot v$ kan på denne måde beregnes ved brug af *tre* multiplikationer, *seks* additioner og *to* subtraktioner af tal med halvt så mange cifre. Derved reduceres det oprindelige problem til tre multiplikationer, idet vi kan se bort fra additionerne og subtraktionerne, da deres kompleksitet er lineær. Metoden leder direkte til en rekursiv implementering, hvor man nedbryder problemet, indtil det består af en-cifrede multiplikationer, som let kan udføres ved brug af de indbyggede aritmetiske funktioner for enkeltpræcisionstal.

Antallet af sådanne multiplikationer der kræves for at multiplicere to n-bits tal, er da givet ved rekursionsformlen

$$M(n) = 3M(n/2) \tag{11.3}$$

da vi, for hver multiplikation, skal udføre tre multiplikationer med halvt så store tal. Hvis vi antager at $n = 2^k$ og at $M(1) = 1$, gælder der følgende :

$$M(2^k) = 3M(2^{k-1}) = 3^2M(2^{k-2}) = \cdots = 3^kM(1) = 3^k \qquad (11.4)$$

Udtrykt ved $n$ bliver dette

$$3^k = (2^{\log_2(3)})^k = (2^k)^{\log_2(3)} = n^{\log_2(3)}$$

Vi kan nu give følgende udtryk for kompleksiteten af multiplikation :

$$M(n) \approx n^{\log_2(3)} \approx n^{1.58}$$

en væsentlig forbedring af $n^2$ metoderne.

### Algoritme 11.4 Hurtig multiplikation

1. Hvis faktorerne er enkeltpræcisionscifre, multipliceres de ved brug af systemets indbyggede multiplikation.
2. Ellers opdeles de to faktorer i to halvdele, i hver deres heltal : $U_0, V_0, U_1, V_1$.
3. Beregn henholdsvis :

$$\begin{aligned} C_1 &= U_1 \cdot V_1 \\ C_2 &= U_0 \cdot V_1 \qquad \text{og} \\ C_3 &= (U_1 - U_0)(V_0 - V_1) \end{aligned}$$

   Med grundtal $g$, bliver resultatet

$$(g^{2n} + g^n)C_1 + (g^n + 1)C_2 + g^nC_3$$

Vi har i vores implementering ikke benyttet denne optimering, da den i praksis viste sig at være 10 gange langsommere end *papir og blyant* metoden. Dette skyldes at proceduren er rekursiv, og dermed opretter syv heltal - fire heltal i opdelingen og tre heltal til henholdsvis $C_1$, $C_2$ og $C_3$ - for hvert rekursionstrin. I stedet benytter vi den først beskrevne algoritme til multiplikation.

I vores implementering af den gemmer vi dog ikke alle mellemresultaterne for til sidst at lægge dem sammen, men foretager denne addition med det samme. Derved sparer vi oprettelse af en del multipræcisionstal.

**Algoritme 11.5 Multiplikation**

1. Lad det største tal være multiplikator.
2. For hvert ciffer i multiplikator gør :
   (a) For hvert ciffer i multiplikand gør :
      i. Beregn produktet af de to cifre i de to faktorer. Adder dette produkt til det allerede beregnede resultat, sammen med en eventuel mente.
      ii. Herefter beregnes menten som
      $$Mente = resultat \;//\; Grundtal$$
      iii. og produktcifferet som
      $$Pcif = \text{MOD } (resultat,\ Grundtal)$$

### 11.1.5 Division og Modulus

Den grundlæggende facilitet i forbindelse med modulusaritmetik er division. Kvotienten mellem to multipræcisionstal bruges for eksempel af algoritmen til beregning af inverse og ændring af grundtal. Resten ved divisionen bruges alle de steder, hvor modulus skal beregnes.

Vi forsøgte selv at konstruere en metode der beregnede modulus, uden at danne en kvotient. Metoden viste sig imidlertid at indeholde en 'skjult' beregning af kvotienten, således at der reelt var tale om en almindelig division. Samtidig var den langsommere end den divisionsalgoritme vi på det tidspunkt *havde* implementeret. Derfor valgte vi at bruge denne divisionsalgoritme, også når vi skulle beregne modulus.

#### Division

Følgende symboler benyttes i dette afsnit :

- $u$ - dividenden
- $v$ - divisoren
- $q$ - kvotienten
- $r$ - resten
- $b$ - grundtallet, som alle tal er repræsenteret i.

Hvis man reflekterer over "papir og blyant" metoden, vil en division foregå, som illustreret i det følgende eksempel :

***Eksempel 11.4. Division med papir og blyant***

*Skal man dividere* 3142 *med* 47, *vil man først udføre divisionen* 314/47, *som giver kvotienten* 6 *og resten* 32. *Dernæst beregnes* 322/47 = 6 *med resten* 40, *hvilket i alt giver kvotienten* 66 *og resten* 40.

```
3142 : 47 = 66
--------------
 314
-282
 ---------
  322
- 282
 ---------
   40

Resultat = 66
Rest     = 40
```

Ud fra eksemplet kan vi se, at division af et $(m+n)$ cifret tal med et $n$ cifret tal kan reduceres til en *række* divisioner af et $(n+1)$-cifret tal $u$ (eventuelt med forreste ciffer lig 0), med et $n$-cifret tal $v$, hvor hvert ciffer $q$ opfylder, at $0 \leq q = \lfloor u/v \rfloor < b$. Da resten $r$ efter hvert skridt er mindre end $v$, benytter vi størrelsen $r \cdot b +$ (næste ciffer) som det nye $u$ i næste division.

Denne metode virker også mere generelt, hvorved divisionsproblemet reduceres til følgende formulering :

*Lad* $u = (u_0 u_1 \ldots u_n)_b$ *og* $v = (v_1 v_2 \ldots v_n)_b$, *være positive tal med grundtallet* $b$ *således at* $u/v < b$. *Find en algoritme til fastlægge* $q = \lfloor u/v \rfloor$.

Vi observerer følgende ud fra vores viden om $u$ og $v$

$$u/v < b \Rightarrow u/b < v$$

hvilket vil sige at

$$\lfloor u/b \rfloor < v$$

som er ensbetydende med at

$$(u_0 u_1 \ldots u_{n-1})_b < (v_1 v_2 \ldots v_n)_b$$

Da $0 \leq r < v$ gælder der desuden at $q \geq (u/v) - 1$.

Kernen i hele denne division er at kunne gætte kvalificeret på værdien af $q$ ud fra de mest betydende cifre i $u$ og $v$ alene. Kald et sådant gæt på $q$ for $\hat{q}$ og lad

$$\hat{q} = \min\left(\left\lfloor \frac{u_0 b + u_1}{v_1} \right\rfloor, b - 1\right) \tag{11.5}$$

Formel 11.5 siger at $\hat{q}$ fås ved at dividere de to første cifre i $u$ med det første ciffer i $v$ og hvis resultatet af dette er $b$ eller mere, erstattes det med $b - 1$.

Denne værdi er som regel en meget god tilnærmelse til den rigtige kvotient, og det kan vises at den aldrig er mindre end $q$ :

**Sætning 11.1** *Hvis $\hat{q}$ beregnes efter formel 11.5 gælder at $\hat{q} \geq q$*

**Bevis :**

Da $q \leq b-1$ er sætningen sand i tilfældet $\hat{q} = b-1$. I alle andre tilfælde har vi at

$$\begin{array}{lcll} \hat{q} & = & \left\lfloor \frac{u_0 b + u_1}{v_1} \right\rfloor & \Rightarrow \\ \hat{q} v_1 & \geq & u_0 b + u_1 & \Rightarrow \\ \hat{q} v_1 & \geq & u_0 b + u_1 - v_1 + 1 & \end{array}$$

Det sidste gælder fordi $v_1 \geq 1$.

Beviset forløber ved at demonstrere, at resten ved subtraktionen $u - \hat{q}v = r$ er mindre $v$. Da $\hat{q}v \geq \hat{q} v_1 b^{n-1}$ er

$$\begin{aligned} u - \hat{q}v &\leq u - \hat{q} v_1 b^{n-1} \\ &\leq u_0 b^n + u_1 b^{n-1} + \ldots + u_n - (u_0 b + u_1 - v_1 + 1) b^{n-1} \\ &= u_2 b^{n-2} + \ldots + u_n + v_1 b^{n-1} - b^{n-1} \\ &< v_1 b^{n-1} \\ &\leq v \end{aligned}$$

og da $u - \hat{q}v \leq v$ må $\hat{q} \geq q$. □

Hvis $v_1$ er rimeligt stor vil $\hat{q}$ være en *meget* god tilnærmelse. Faktisk gælder det at

**Sætning 11.2** *Hvis* $v_1 \geq \left\lfloor \frac{b}{2} \right\rfloor$ *så ligger* $q$ *i intervallet*

$$[\hat{q} - 2, \hat{q}]$$

**Bevis :**

Vi *har* vist at $q \leq \hat{q}$. Vi antager i det følgende at $\hat{q} \geq q + 3$, dvs. en modstrid til sætningen, og får så :

$$\hat{q} \leq \frac{u_0 b + u_1}{v_1} = \frac{u_0 b^n + u_1 b^{n-1}}{v_1 b^{n-1}} \leq \frac{u}{v_1 b^{n-1}} \leq \frac{u}{v - b^{n-1}}$$

Den sidste ulighed skyldes at $v_1 b^{n-1} + b^{n-1} > v$. En eventuel division med nul på grund af $v = b^{n-1}$ er umulig her, da

$$v = b^{n-1} = (100\ldots0)_b \Rightarrow q = \hat{q}$$

hvilket strider imod antagelsen.

Da vi har observeret at $q > (u/v) - 1$ følger det at

$$\begin{aligned} 3 &\leq \hat{q} - q \\ &< \frac{u}{v - b^{n-1}} - (\frac{u}{v} - 1) \\ &= \frac{uv - u(v - b^{n-1})}{v(v - b^{n-1})} + 1 \\ &= \frac{u}{v}\left(\frac{b^{n-1}}{v - b^{n-1}}\right) + 1 \end{aligned}$$

Derfor er

$$\frac{u}{v} > 2\left(\frac{v - b^{n-1}}{b^{n-1}}\right) \geq 2(v_1 - 1)$$

Endelig får vi at

$$\begin{aligned} b-1 &\geq \hat{q} \geq q+3 \\ b-4 &\geq \hat{q}-3 \geq q \text{ og da } q = \lfloor u/v \rfloor \\ b-4 &\geq q = \lfloor u/v \rfloor \geq 2(v_1-1) \\ b-4 &\geq 2v_1-2 \end{aligned}$$

$$\frac{b}{2}-1 \geq v_1$$

$$\left\lfloor \frac{b}{2} \right\rfloor > v_1$$

Det betyder at hvis $\hat{q} \geq q+3$, må $v_1$ være mindre end eller lig $\lfloor \frac{b}{2} \rfloor$. □

Det vigtigste ved sætning 11.2 er, at uanset hvor stor $b$ er, så vil den gættede kvotient $\hat{q}$ aldrig være mere end 2 for stor. Betingelsen for dette ($v_1 \geq \frac{b}{2}$) kan opnås ved at multiplicere både $u$ og $v$ med et tal, der normaliserer forholdet mellem dem. Værdien af kvotienten ændres *ikke* ved dette og resten kan genskabes ved at dividere $u$ med denne normaliseringsfaktor efter at divisionen er færdig. Normaliseringsfaktoren kan vælges som $(\frac{b}{v_1+1})$ i de tilfælde, hvor $v_1 < \frac{b}{2}$.

Inden vi beskriver algoritmen viser vi at valget af $\hat{q}$ kan forbedres ved at tage lidt flere cifre i $u$ og $v$ i betragtning, så den højst er 1 for stor. Dette foregår som følger :

Sålænge

$$(v_1 b + v_2)\hat{q} > u_0 b^2 + u_1 b + u_2$$

er $\hat{q}$ for stor, og der subtraheres 1.

Når subtraktionen $u := u - \hat{q}v$, udføres kan et eventuelt forkert resultatet korrigeres med 1 addition : $u := u + v$. Dette gør algoritmen hurtig idet alle cifrene kun er i spil i en multiplikation og højst to additioner.

Algoritmen antager at

- $(u, v) \in \mathbb{N}_+$
- $u > v$

- $v$ har mindst 2 cifre

**Algoritme 11.6 Division**

1. Hvis $v_1 < \frac{b}{2}$ så normer $u$ og $v$. Antallet af cifre i $v$ ændres ikke af dette. Algoritmen forventer derimod, at $u$ har fået et ekstra ciffer efter normaliseringen. Er dette ikke direkte tilfældet tilføjes et ekstra nul forrest i $u$.
2. Sålænge $u$ har flere cifre end $v$ så læg $u$'s sidste ciffer på en stak.
3. Sålænge der er mere på stakken udfør følgende
   (a) Tag øverste ciffer fra stakken og indsæt det sidst i $u$
   (b) Sæt $q := \min(\lfloor (u_0 b + u_1)/v_1 \rfloor, b-1)$
   (c) Sålænge $v_2 q < (u_0 b + u_1 - q v_1) b + u_3$ lad $q := q - 1$
   (d) Lad $u := u - qv$
   (e) Hvis $u < 0$ lad $u := u + v$ og lad $q := q - 1$
   (f) Indsæt $u_0$ forrest i $r$. Omdøb cifrene i $u$ så det forreste hedder $u_0$.
   (g) Indsæt $q$ forrest i Kvotient-variablen.
4. Indsæt resten af $u$ forrest i $r$
5. Hvis der blev normaliseret så afnormer $r$.

### 11.1.6 Modulær Eksponentiering

Modulær eksponentiering, beregner funktionen :

$$y = x^k \text{ MOD } n \quad k \geq 0 \quad n > 0$$

I talteorikapitlet beskrev vi hvordan funktionen kunne udregnes rimelig let, ved brug af multiplikationer, kvadreringer og beregninger af modulus. Denne beskrivelse leder direkte frem til den følgende rekursive algoritme :

**Algoritme 11.7 Expomod**

1. Hvis eksponenten er 0, så returner 1.

2. Hvis eksponenten er lige, så bliver resultatet

$$\text{Expomod}\left(x, \frac{k}{2}, n\right)^2 \text{ MOD } N$$

3. Hvis eksponenten er ulige bliver resultatet :

$$x \cdot \text{Expomod}(x, k-1, N) \text{ MOD } n$$

### 11.1.7 Største fælles divisor og Euclids modificerede algoritme

**Største fælles divisor**

Den største fælles divisor beregnes som beskrevet i afsnit 2.3.2 :

**Algoritme 11.8 Største fælles divisor**

1. Hvis $v = 0$ så returner $u$
2. ellers kald funktionen rekursivt med $\text{SFD}(v, u \text{ MOD } v)$

**Euclids modificerede algoritme**

Ligningen

$$u \cdot x \equiv 1 \bmod v \tag{11.6}$$

løses med Euclids modificerede algoritme, beskrevet i afsnit 2.3.2.

Lad

$$\text{SFD}(u, v) = u_3 = uu_1 + vu_2$$

Så vil den modificerede algoritme i hvert gennemløb

$$\begin{aligned} uu_1 + vu_2 &= u_3 = u \text{ MOD } v \\ uv_1 + vv_2 &= v_3 \end{aligned}$$

og derudfra få

$$ut_1 + vt_2 = t_3$$

hvor

$$\begin{aligned} t_1 &= u_1 - v_1(u_3//v_3) \\ t_2 &= u_2 - v_2(u_3//v_3) \\ t_3 &= u_3 - v_3(u_3//v_3) = u_3 \text{ MOD } v_3 \end{aligned}$$

***Eksempel 11.5. Beregning af inverse***

*Skal vi i ligningen*

$$13 \cdot x \equiv 1 \bmod 47$$

*finde den inverse $x$, foregår det som følger. Vi sikrer os at* $\text{SFD}(13, 47) = 1$. *Vi sætter*

$$(u_1, u_2, u_3) := (1, 0, 13) \text{ og } (v_1, v_2, v_3) := (0, 1, 47)$$

*Algoritmen reducerer da på ligningerne :*

$$\begin{aligned} u_1 \cdot u + u_2 \cdot v &= u_3 \\ v_1 \cdot u + v_2 \cdot v &= v_3 \end{aligned}$$

*ved hele tiden at subtrahere et multiplum af forholdet mellem ligningerne og den nederste fra den øverste :*

$$\begin{array}{rcrcll} 1 \cdot 13 & + & 0 \cdot 47 & = & 13 & \\ 0 \cdot 13 & + & 1 \cdot 47 & = & 47 & q = 0 \\ 1 \cdot 13 & + & 0 \cdot 47 & = & 13 & q = 3 \\ -3 \cdot 13 & + & 1 \cdot 47 & = & 8 & q = 1 \\ 4 \cdot 13 & + & (-1) \cdot 47 & = & 5 & q = 1 \\ -7 \cdot 13 & + & 2 \cdot 47 & = & 3 & q = 1 \\ 11 \cdot 13 & + & (-3) \cdot 47 & = & 2 & q = 1 \\ -18 \cdot 13 & + & 5 \cdot 47 & = & 1 & q = 1 \end{array}$$

*Den inverse til* 13 *modulo* 47 *er således* $-18$.

I algoritmen antages det at $\text{SFD}(u, v)$ er 1.

**Algoritme 11.9 Beregning af inverse**

1. Initialiser de variable således at

$$(u_1, u_2, u_3) = (1, 0, u) \text{ og } (v_1, v_2, v_3) = (0, 1, v)$$

2. Sålænge $v_3 \neq 0$ gør :
   (a) Sæt $q = \lfloor \frac{u_3}{v_3} \rfloor$
   (b) Beregn de nye værdier for $t$ :

$$(t_1, t_2, t_3) := (u_1, u_2, u_3) - (v_1, v_2, v_3) \cdot q$$

   (c) Sæt de nye værdier ind i hhv $u$ og $v$

$$(u_1, u_2, u_3) := (v_1, v_2, v_3)$$

$$(v_1, v_2, v_3) := (t_1, t_2, t_3)$$

3. Den inverse er nu $u_1$.

### 11.1.8 Sammenligninger

Vi har implementeret de tre grundlæggende sammenligningsoperatorer, større end, mindre og lig med.

De tre sammenligningsprocedurerne er opbygget, så de først tester på antallet af cifre i tallene. Er de ens sammenlignes størrelsen af de enkelte cifre forfra i tallet. Algoritmen for $x$ mindre end $y$ er som følger.

#### Algoritme 11.10 Mindre end

1. Hvis antallet af cifre i $x$ er mindre end antallet af cifre i $y$, så er sammenligningen sand.
2. Hvis antallet af cifre i $x$ er større end antallet af cifre i $y$, er sammenligningen falsk.
3. Hvis antallet af cifre i $x$ er lig med antallet af cifre i $y$, så fortsæt med at sammenligne de enkelte cifre i tallene forfra, indtil der er 2 der er forskellige.
4. Hvis $x$ ciffer på et eller andet tidspunkt er mindre end det tilsvarende ciffer i $y$, så er sammenligningen sand, ellers er den falsk.

## 11.2 RSA-Værktøjet

Efter at have beskrevet de forskellige funktioner i multipræcisionsværktøjet, vil vi gennemgå selve RSA-algoritmerne. Vi har tidligere beskrevet teorien *bag* algoritmerne, og vil i dette kapitel derfor kun tilføje de detaljer, der er af praktisk betydning for systemets funktion.

### 11.2.1 Primtalstest

Generering af primtal, forudsætter en primtalstest. Vi har valgt at benytte Miller-Rabins stokastiske primtalstest, som vi har beskrevet i detaljer i kapitel om faktorisering og primtalstest.

Algoritmen returnerer falsk når det tal, der testes viser sig at være sammensat og sand, hvis det ikke gør det. For at optimere algoritmen har vi en gang for alle, dannet den tabel med testbaser, som algoritmen bruger.

**Algoritme 11.11 Miller-Rabin**

Bemærk at alle beregninger forgår modulus $h$.

1. Split $h-1$ i komponenterne $s$ og $t$ så $h-1 = 2^s t$, og $t$ ulige.
2. Sålænge testen ikke afslører $h$ som sammensat og $k \leq$ testantal gør :
   (a) $b :=$ Næste base.
   (b) Hvis $b^{h-1} \neq 1$ eller $\mathrm{SFD}(b,h) \neq 1$ så er h sammensat ellers fortsæt.
   (c) Lad $b := b^t$
   (d) Hvis $b = 1$ eller $b = -1$ er $h$ muligvis et primtal og der fortsættes med næste base.
   (e) Ellers : Sålænge $b \neq -1$ og $b \neq 1$ gør $b := b^2$
   (f) Hvis $b = 1$ er $h$ sammensat
   (g) ellers fortsættes med næste base.

### 11.2.2 Stærke Primtal

Funktionen generer et stærkt primtal, der opfylder de betingelser vi gav i kapitlet om sikkerhed i RSA-kryptosystemet.

**Algoritme 11.12 Stærke primtal**

1. generer primtal $s$ og $t > \mathcal{M}$(sæd, blokstørrelse), hvor $\mathcal{M}$ er en funktion, der angiver minimum for $s$ og $t$.
2. Sålænge $r$ ej primtal gør : Lad $r := 2lt + 1$. Lad $l := l + 1$.
3. Beregn $P_0 := s^{r-1} - r^{s-1}$
4. Hvis $P_0$ er lige, så lad $P_0 := rs$
5. Sålænge $p$ ej primtal gør : Lad $p := P_0 + 2krs$. Lad $k := k + 1$

### 11.2.3 Nøgler

Nøglegeneratorens funktion er at generere de to nøglesæt den offentlige nøgle $(e, n)$ og den hemmelige nøgle $(d, n)$.

Funktionen er opdelt i to : En som genererer et nøglesæt til brugeren, men ikke returnerer de genererede primtal $p$ og $q$ og en, som desuden returnerer $p$ og $q$. Det er i den sidstnævnt funktion, at nøglerne faktisk oprettes. Nøglegenereringen foregår ud fra et sikkerhedsniveau, som er det mindste antal decimale cifre der må være i $N$.

**Algoritme 11.13 Nøglegenerering**

1. generer $p$ som et stærkt primtal.
2. generer $q$ som et stærkt primtal.
3. Lad $N := p \cdot q$.
4. Lad $\phi(N) := (p-1) \cdot (q-1)$
5. vælg et $d$ så $\mathrm{SFD}(e, \phi(N)) = 1$ og $d > \max(p, q)$
6. Lad $e := d^{-1} \bmod \phi(N)$

### 11.2.4 Konvertering mellem ASCII-tegn og decimaltekster

Konverteringen fra ASCII tegn til en repræsentation i multipræcisionstal foregår efter samme mønster som konverteringen fra decimaltekst til multipræcisionstal, i to omgange. Da talværdierne for ASCII-tegn ligger i intervallet $[0, 127]$, kan *et* ASCII-tegn repræsenteres som *et* ciffer i et multipræcisionstal med grundtal 128. Vi starter derfor med at indlæse ASCII-teksten i et sådant multipræcisionstal. Derefter konverteres dette multipræcisionstal til det ønskede grundtal.

Algoritmen er som følger :

**Algoritme 11.14 Konverter**

1. Opret et tomt multipræcisionstal med grundtal 128.
2. Sålænge der er flere tegn i teksten gør :
   (a) Tag det næste tegn og find dets ASCII-værdi.
   (b) Indsæt denne værdi som et ciffer i multipræcisionstallet.
3. Udskriv det oprettede multipræcisionstal som decimaltekst, med procedure `Multital_ud`.

Konvertering tilbage til ASCII-tekst foregår helt analogt ved at skifte multipræcisionstallet til grundtal 128 og indsætte cifrene som ASCII-tegn i en tekst.

### 11.2.5 Cifferblokkobling

Cifferblokkobling implementeres normalt ved med en exclusiv-or funktion på de binære data, da dette giver identiske funktioner for koblingen og afkoblingen. Da det ikke umiddelbart er muligt at foretage exclusiv-or i SIMULA, og da det ikke specielt hurtigt i et højniveau-sprog, har vi valgt ikke at beregne CBK på denne måde. I stedet benytter vi addition og subtraktion i restklassegruppen 10 (grundtallet for de enkelte cifre), på de konvertede decimaltekster. Denne funktion har de samme egenskaber som exclusiv-or, og lader sig lettere beregne. Til gengæld er vi så nødt til at have to funktioner, der er hinandens inverse : `CBK_ind` og `CBK_ud`. CBK-funktionerne i programmet, arbejder kun en enkelt blok

ad gangen. Det er så op til det kaldende program at styre opdelingen i blokke.

### *Eksempel 11.6. CifferBlokKobling*

*Hvis der skal laves CBK-ind på de følgende to tekster,*

```
123457678964656586
563523535645675675
```

*foregår det således :*

```
  123457678964656586
'+' 563523535645675675
    --------------------
  686970103509221151
```

*CBK-ud foregår tilsvarende :*

```
  686970103509221151
'-' 563523535645675675
    --------------------
  123457678964656586
```

### CBK-ind

Inddata er en klartekstblok og en ciffertekstblok.

### Algoritme 11.15 CBK_ind

1. Der indsættes nuller i den korteste blok, så de bliver lige lange.
2. For alle tegn i teksten gør
   (a) Omdan de to tegn til integers og adder dem modulo 10.
   (b) Omdan resultatet til et tegn og indsæt det i resultatteksten.

### CBK-ud

Denne algoritme fungerer fuldstændig efter samme mønster som CBK-ind, blot anvender den subtraktion modulo 10, for at komme tilbage til det oprindelige. Det er i denne forbindelse vigtigt at bemærke at rækkefølgen af inddataparametre *ikke* er ligegyldig.

## 11.2.6 Ind- og afkodning

Med de ovenstående rutiner til rådighed, er ind- og afkodning lige ud af landevejen :

### Algoritme 11.16 Indkodning

1. Konverter nøglerne fra decimaltekster til multipræcisionstal.
2. Konverter klarteksten til decimaltekst.
3. Del klarteksten op i klartekstblokke.
4. For hver *klartekstblok* gør :
   (a) Hvis CBK er valgt :
   *mellemblok* := CBK *(klartekstblok, forrige ciffertekst-blok)*
   (b) ellers
   *mellemblok* = *klartekstblok*
   (c) Konverter *mellemblok* til multipræcisionstal.
   (d) Indkod ved

$$\textit{ciffertekstblok} := \textit{mellemblok}^e \text{ MOD } n$$

   (e) Konverter cifferteksten fra multipræcisionstal til decimaltekst.
5. Saml ciffertekstblokkene til cifferteksten.
6. Konverter cifferteksten fra decimaltekst til almindelig tekst.

Skulle klarteksten ikke bestå af et helt antal blokke, laves de overskydende tegn til en selvstændig blok.

Afkodning foregår efter samme mønster.

**Algoritme 11.17 Afkodning**

1. Konverter nøglerne fra decimaltekster til multipræcisionstal.
2. Konverter cifferteksten til decimaltekst.
3. Del cifferteksten op i ciffertekstblokke.
4. For hver *ciffertekstblok* gør :
   (a) Konverter ciffertekstblokken til multipræcisionstal.
   (b) afkod ved
   $$mellemblok := ciffertekstblok^e \text{ MOD } n$$
   (c) Konverter mellemblokken fra multipræcisionstal til decimaltekst.
   (d) Hvis CBK er valgt :
   *klartekst* := CBK (*mellemblok, forrige ciffertekstblok*)
   (e) ellers
   *klartekstblok:* = *mellemblok*
5. Saml klartekstblokkene til klarteksten.
6. Konverter klarteksten fra decimaltekst til almindelig tekst.

Bemærk at rækkefølgen af CBK og kodning ikke er den samme i de to algoritmer.

## 11.3 Test af programmet

### 11.3.1 Overordnet teststrategi

Vi har testet vores program ud fra den strategi, at vi først tester de funktioner, der ikke er afhængige af andre funktioner. Så kan disse antages at fungere korrekt når de overliggende funktioner skal testes. Dog er der nogle af funktionerne, der er sidestillede, f.eks subtraktion og addition. Dette skyldes, at de begge kalder hinanden ved nogle bestemte kombinationer af inddata f.ex er $-3 + 4 = 4 - 3$.

Selve beregningsresultaterne er testet ved brug af lommeregner og hovedregning, og for nogle af multipræcisionstalsprocedurerne er rigtigheden af procedurerne verificeret ved brug af et BASIC program (U-BASIC) til beregning med store tal og primtalstest. Derudover har vi benyttet følgende multiplikationsregel, hentet fra [30] :

Der gælder for alle $m < n$ og grundtal $t$ at :

$$\overbrace{(t^m-1)\cdot(t^n-1)}^{Produkt} = \overbrace{\underbrace{(t-1)\ldots(t-1)}_{m-1}(t-2)\underbrace{(t-1)\ldots(t-1)}_{n-m}\underbrace{(0)\ldots(0)}_{m-1}(1)}^{Cifre}$$

hvor (0) er nul i det grundtal, og (1) tilsvarende er en. Hvis grundtallet for eksempel er 10, får man :

$$(10^4-1)(10^9-1) = 9.998.999.990.001$$

### Ekstern test

En ekstern test er en systematisk afprøvning af alle mulige kombinationer af inddata. Da dette ikke kan lade sig gøre med et program, der har alle naturlige tal som inddata, deler vi inddata op i nogle ækvivalensklasser og tester funktionen med en repræsentant for hver af disse.

For os er de interessante grupper, positive og negative tal, nul, samt encifrede og flercifrede tal. I vores arbejde med tal med et givet grundtal, betyder dette, at vi skal afprøve kombinationer med 0, et enkelt ciffer (hvis grundtallet er 10.000 kan det være 7513), et negativt enkelt ciffer, et flercifret tal og et negativt flercifret tal.

### Intern test

En intern test er en test, der afprøver alle forgreninger i hver enkelt procedure. Dette er nødvendigt, da ikke alle grænsemuligheder vil blive checket, ved at undersøge alle de ovenfor beskrevne typer af inddata.

### Testpraksis

Dels for at kunne teste programmet og dels for at kunne afprøve det, har vi lavet en testmenu. I denne har vi opstillet alle procedurer, så den enkleste har nr 1, og den der anvender flest andre procedurer er placeret sidst. En test af programmet består så i at vælge alle punkter i rækkefølge med de rigtige kombinationer af inddata. Disse inddata har vi lagret på fil, en for hver procedure.

### 11.3.2 Test af Procedure Div

Procedure Div er vel nok den mest komplicerede procedure i programmet Vi har derfor valgt at tage den ud som et eksempel på hvorledes vi har testet programmet.

#### Ekstern test

Der er de følgende mulige kombinationer af inddata :

$$\{0, \pm enkeltpræcisionstal, \pm multipræcisionstal\} \times \{0, \pm enkeltpræcisionstal, \pm multipræcisionstal\}$$

Dette giver de 25 tests i tabel 11.2.

Når divisionsalgoritmen er checket for alle disse inddatatyper, er den eksterne test ok.

#### Intern test

Proceduren giver de følgende interne tests (sammenlign eventuelt med udskriften af proceduren i appendixA :

1. Dividenden er negativ og divisoren er positiv - linje 41. - Dette tilfælde er repræsenteret i den eksterne test.
2. Dividenden er positiv og divisoren er negativ - linje 52. - Dette tilfælde er repræsenteret i den eksterne test.
3. Både dividend og divisor er negative - linje 63. - Dette tilfælde er repræsenteret i den eksterne test.
4. Divisor er nul - linje 72. - Dette tilfælde er repræsenteret i den eksterne test.
5. Divisor = dividend - linje 74. - Dette tilfælde er ikke nødvendigvis tilstede ved den eksterne test, og skal derfor testes eksplicit med f.eks.

| Nr : | Divisor | Dividend |
|---|---|---|
| 26 | 1234 5678 | 1234 5678 |

| Nr | Divisor | Dividend | Inddataklasse |
|---|---|---|---|
| 1 | 0 | 0 | $0 \times 0$ |
| 2 | 0 | $E$ | $0 \times \pm E$ |
| 3 | 0 | $-E$ | |
| 4 | $E$ | 0 | $\pm E \times 0$ |
| 5 | $-E$ | 0 | |
| 6 | 0 | $M$ | $0 \times \pm M$ |
| 7 | 0 | $-M$ | |
| 8 | $M$ | 0 | $\pm M \times 0$ |
| 9 | $-M$ | 0 | |
| 10 | $M$ | $E$ | $\pm M \times \pm E$ |
| 11 | $M$ | $-E$ | |
| 12 | $-M$ | $E$ | |
| 13 | $-M$ | $-E$ | |
| 14 | $E$ | $M$ | $\pm E \times \pm M$ |
| 15 | $E$ | $-M$ | |
| 16 | $-E$ | $M$ | |
| 17 | $-E$ | $-M$ | |
| 18 | $E$ | $E$ | $\pm E \times \pm E$ |
| 19 | $E$ | $-E$ | |
| 20 | $-E$ | $E$ | |
| 21 | $-E$ | $-E$ | |
| 22 | $M$ | $M$ | $\pm M \times \pm M$ |
| 23 | $M$ | $-M$ | |
| 24 | $-M$ | $M$ | |
| 25 | $-M$ | $-M$ | |

Her betegner E : Et enkeltcifret tal
M : Et flercifret tal

**Tabel 11.2:** Mulige inddatakombinationer for procedure Div

6. Dividend mindre end divisor - linje 80. Dette tilfælde er dækket ind af den eksterne test.

7. Tilfældet, hvor divisoren har mindre end to cifre - linje 86, er dækket ind i den eksterne test.

8. Både dividend og divisor er positive flercifrede multipræcisionstal og divisor er mindre end dividend - linje 91.

   Dette er den største del af proceduren. Den giver anledning til de følgende testtilfælde :

   (a) Testen falder i to dele :

   Enten er det første ciffer i divisoren større end *grundtal*//2. I dette tilfælde sker der ingenting, og det er derfor ikke så interessant at teste.

   Ellers er det første ciffer mindre end *grundtal*//2 - linje 104. Så skal vi teste at forgreningen reagerer som ønsket. Dette tilfælde testes ikke automatisk af de 25 eksterne tests. Vi må derfor opstille en eksplicit test, der tager højde for dette. for eksempel for *grundtal* = 10000.

| Nr : | Divisor | Dividend |
|---|---|---|
| 27 | 3219 9993 | flercifret tal |

   (b) Næste punkt er et check på om længden af dividenden efter den eventuelle normalisering er lig længden af den oprindelige dividend - linje 104. For at checke denne længde, kan følgende test opstilles.

| Nr : | Divisor | Dividend |
|---|---|---|
| 28 | 3219 993 | 3436 6547 7687 |
| 29 | 3219 993 | 2435 2344 |

   (c) Vi skal nu teste en essentiel forgrening, nemlig den løkke hvor selve divisionen foregår - linje 141. Den løkke vil altid gennemløbes mindst en gang. Denne test er derfor dækket ind af de eksterne testdata.

   (d) Forgreningen, linje 152 skal afsløre, om det første tal i divisoren er lig det første tal i dividenden. Den er meget svær

at teste. Man fanger den kun ved at køre løkken flere gange igennem, da det aldrig vil være tilfældet første gang vi passerer denne forgrening. Vi har fundet nogle passende testdata, der afslører dette.

| Nr : | Divisor |
|---|---|
| 30 | 9999 9999 9999 |

| Dividend |
|---|
| 9999 9999 9998 9999 9999 0000 0000 0001 |

(e) Den næste forgrening udføres hvis kvotienten er blevet for stor - linje 155. Det er et tilfælde, som næsten aldrig opstår og som derfor meget vanskeligt kan testes. Vi har konstrueret følgende testeksempel :

| Nr : | Divisor | Dividend |
|---|---|---|
| 31 | 1000 0000 0000 | 5000 9999 |

(f) Test nr 30 vil også sørge for at den næste forgrening, hvor dividenden er negativ - linje 161, bliver udført.

(g) Forgreningen i linje 169, vil blive testet med de opstillede testdata til den eksterne test.

(h) Hvis divisionen ikke går op, vil resten fra divisionen være i dividenden - linje 176. Om dette tilfælde er dækket af de eksterne test, kan vi ikke garantere, men sandsynligheden er stor. Hvis tilfældet ikke er dækket, kan den testes med :

| Nr : | Divisor | Dividend |
|---|---|---|
| 33 | 1234 5678 | 3453 3332 |

(i) Det sidste der skal testes, er det tilfælde, hvor det første ciffer af divisoren er mindre end *grundtal*//2, og det har vi allerede testet.

## 11.4 Vurdering af implementationen

Først og fremmest må vi sige, at implementationen ikke er blevet helt færdig. Der er endnu nogle afkroge af programmet der ikke opfører sig helt som de burde, og der er nogle enkelte dele af designet der ikke er implementeret. De basale ting - multipræcisionsaritmetik og primtalstest fungerer tilfredsstillende i henhold til de udførte tests.

Derudover er det interessante den hastighed som programmet kører med. Indkodning af en klartekstblok på 80 bogstaver, med en nøgle $n$ på 200 cifre og $e$ på 100 cifre, tager cirka 4 minutters cpu-tid. Dette er naturligvis fuldstændig uanvendelige tider i praksis. Skal man f.eks. indkode 1 tekstside pr minut, skal der indkodes 1 blok hvert andet sekund. Skal man holde takt med et modem der sender 9600 bits/sekund skal 1 blok indkodes på ca. 0.06 sekund. I forhold til vores implementeringen skal der arbejdes henholdsvis 100 og 3000 gange hurtigere. Da vi ikke benytter specielt langsomme algoritmer, er det klart at de nødvendige hastigheder ikke kan opnås i SIMULA. Selv en implementering i C eller assembler vil have store problemer med at følge med. Det er da også sådan at alle kommercielle implementeringer i dag, foregår i hardware, for at det kan gå hurtigt nok. Eksempler er [47], [48] og [32].

# Kapitel 12

# Anvendelse af RSA-kryptosystemet

Vi vil nu give to eksempler på, hvordan RSA kan anvendes i praksis. Først beskriver vi designet af krypteringsdelen af et elektronisk post-system, byggende på vores RSA-værktøj. Derefter beskriver vi, hvordan RSA anvendes i DANKORT-systemet.

## 12.1 RSA i et elektronisk post system

I dag sendes der flere og flere vigtige dokumenter med elektronisk post. Derfor stiger behovet for elektroniske post systemer, der kan beskytte de data, der sendes rundt. Da almindelige offentlige netværk ikke yder nogen beskyttelse mod aflytning, er der naturligt et behov for kryptografi. Hvis hver person i post-systemet ønsker at beskytte sine data, bliver antallet af nøgler så stort, at et hemmelig-nøgle system ikke er anvendeligt. Det er derfor oplagt at bruge et offentlig-nøgle system, som f.eks. RSA.

Det er ikke vores mål at beskrive designet af et komplet system. Vi vil koncentrere os om krypteringsdelen, og antage at resten af systemet findes i forvejen, og opfører sig som et normalt elektronisk post-system, som f.eks. RUC's mail system, der indeholder følgende funktioner :

- Send besked.
- Læs modtaget besked.

- Slet besked.

I vores design af udvidelsen af et elektronisk post system, vil vi dels antage eksistensen af et RSA-værktøj, som det vi har beskrevet i de foregående kapitler, og dels at systemet skal køre på en traditionel flerbrugermaskine, uden speciel krypterings-hardware.

### 12.1.1 Grænsefladen

Udgangspunktet for vores design er, at det skal være så enkelt som muligt. Man bør ikke 'forstyrre' den almindelige bruger, som ikke ønsker at anvende kryptografi, med de funktioner der skal benyttes til kryptering. Samtidig bør man heller ikke belemre kryptografi-brugeren med viden om det anvendte kryptosystem. Fra brugerens synspunkt er det ligegyldigt, hvilket system, der er implementeret - om det er RSA-systemet eller et andet offentlig nøgle system. Det skal blot være et sikkert system.

De funktioner systemet skal udføre er :

- Nøglegenerering.
- Indkod og send meddelelse.
- Afkod og læs meddelelse.

For ikke at gøre den almindelige del af post-systemet, afsending m.m. uden kryptografi mere besværlig, bør disse krypteringsfunktionerne lægges som selvstændige ordrer i systemet og ikke som en valgmulighed i de almindelige ordrer. Indkod og afkod bør også henholdsvis sende og læse meddelelsen, så brugeren ikke først skal indkode og derefter sende. Det vil være for omstændeligt.

Når de enkelte brugeres nøgler skal genereres, bør man sørge for, at hver enkelt bruger kun har mulighed for at få et sæt nøgler. Dette er vigtigt, da det ellers vil være problematisk at identificere, hvem man egentlig skal sende sit brev til. Det er nemlig kompliceret at skulle fastlægge flere nøgler til det samme brugernavn, da vi forestiller os, at modtageren at et brev specificeres ved hendes brugernavn, når det afsendes. På den anden side bør brugeren have mulighed for at ændre

sin nøgle, hvis han ønsker det. Dette er væsentligt, da det muliggør opretholdelsen af et vist sikkerhedsniveau.

I vore RSA-værktøj findes der, som beskrevet i kapitel 10 en række muligheder for at justere systemet f.eks om CBK ønskes eller ej, eller hvor stort sikkerhedsniveauet skal være. For at gøre systemet så enkelt som muligt, bør alle disse ting være lagt fast på forhånd, så brugeren ikke skal bekymre sig om dem. Man kan på den anden side let forestille sig, at der er behov for at ændre på nogle af de givne parametre i forbindelse med post til eller fra 'fremmede' systemer. Hvis der eksisterer et sådant behov, kan det eventuelt imødekommes, hvis der sammen med hver offentlig nøgle gemmes en brugerprofil, der beskriver hvordan ind- og afkodning skal foregå. En sådan profil kan konstrueres, så der er skjult for den almindelige bruger.

Blandt de valgmuligheder der findes i RSA-værktøjet, har vi i vores tænkte post-system valgt at gøre signering af meddelelser og CBK obligatorisk, fordi det gør systemet mere sikkert. For de øvrige parametre har vi valgt de prædefinerede værdier fra RSA-værktøjet.

Inden vi diskuterer grænsefladen færdig vil vi se på de øvrige sider af systemet.

### 12.1.2 Nøgleadministration

Nøgleadministration er et punkt, som vi bevidst ikke har taget højde for i vores RSA-værktøj. Nøgleadministration deler sig skarpt i to halvdele : Administration af de offentlige nøgler og administration af de hemmelige nøgler.

#### Administration af offentlige nøgler

Det vigtigste for de offentlige nøgler er, at de er tilgængelige og korrekte.

Man kan forestille sig at nøglerne kommer ind i systemet på to måder. Enten får systemet dem fra et andet system. Sådanne nøgler bør være kontrollerede (dvs. autentiske) og godkendte. Hvis disse nøgler er godkendte, kan folk 'stole' på dem, og de kan derfor gøres offentligt tilgængelige. En anden mulighed er, at en bruger personligt får en nøgle fra en bekendt eller kollega. En sådan nøgle er ikke godkendt, da man ikke kan være sikker på dens korrekthed. Den bør derfor ikke gøres tilgængelig for alle, det ville gøre det for let at snyde med nøglerne.

Hvis det er en nøgle til en vigtig person, kan systemet sørge for at den bliver kontrolleret og godkendt.

For ikke at forhindre brugerne i at kommunikere sikkert med nogen, bør systemet kunne håndtere begge nøgleadministrations situationerne. Dette kan gøres ved at anvende et dobbelt system af nøgletabeller. Der skal dels være en global tabel, som administreres af systemet, hvor alle de godkendte og kontrollerede brugere og deres nøgler står. Derudover kan hver bruger konstruere en personlig nøgletabel, hvor han kan gemme de ikke godkendte nøgler, han ønsker at benytte. Når systemet skal indkode en meddelelse til en bruger, leder det så de to tabeller igennem efter tur, for at finde nøglen. I sådanne tabeller kan den enkelte brugers profil iøvrigt også gemmes.

For at dette system kan administreres, kræves der et administrationsværktøj. Med det skal man kunne oprette, slette og rette i hver brugerindgang i de to tabeller. De to tabeller bør være beskyttet således, at det kun er systemet henholdsvis brugeren der kan rette i dem. Alle bør kunne læse i system-tabellen, da det er her de kendte og korrekte nøgler skal hentes.

### Administration af de hemmelige nøgler

Hver bruger har en hemmelig nøgle, som skal opbevares så ingen andre kan få fat i den.

En mulighed for dette er, at bruge en hovednøgle til at kryptere dem. Dette kræver imidlertid at hovednøglen kan gemmes sikkert, hvad der ikke er muligt i et almindeligt edb-system.

En anden mulighed var at lade den enkelte bruger huske sin egen hemmelige nøgle således, at den overhovedet ikke ligger på systemet. En RSA-nøgle er imidlertid på ca. 660 bits, hvilket svarer til ca. 90 ASCII-tegn. Det må formodes at brugeren vil have svært ved at huske 90 fuldstændig tilfældige tegn, og også kunne indtaste dem korrekt. Derfor er denne løsning temmelig upraktisk.

Det vi har valgt at gøre, er at bruge den sidste løsningsmulighed i en modificeret udgave. I stedet for at lade brugeren huske den lange RSA-nøgle, indkoder vi RSA-nøglen med et andet kryptosystem, hvis nøgler er kortere og dermed lettere at huske. Det vil typisk være ved brug af hemmelignøgle systemet DES. I dette system er nøglen 56 bit lang, og kan vælges frit. Det betyder, at vi kan lade brugeren vælge

sin egen DES-nøgle, modsat RSA hvor systemet skal vælge nøglerne på baggrund af kravene for nøglernes udseende. Hvis vi lod brugeren vælge en 8 tegn lang ASCII-streng, ville de fleste vælge en meget enkel streng. Dette ville mindske antallet af tænkelige nøgler en kryptoanalytiker skulle prøve igennem, og dermed øge usikkerheden i systemet.

For at undgå denne faktor lader vi brugeren vælge en 15-20 tegns tekst, og konstruerer ud fra denne en DES-nøgle. Et sidste problem mangler endnu at blive løst. Da alle 56 bits strenge er korrekte DES-nøgler, kan brugeren nemt taste en forkert nøgle ind. Derfor gemmer vi DES-nøglen på systemet indkodet med en envejsfunktion således, at vi kan verificere den, når den tastes ind.

### 12.1.3 Sikkerhed iøvrigt

Da programmet skal køre på en almindelig flerbruger maskine, er der nogle af sikkerhedsproblemerne, som bliver vanskelige at løse tilfredsstillende.

Først og fremmest er der ingen sikkerhed på linien mellem terminal og maskine. Det betyder, at nøglerne kommer til at passere denne linie i klartekst, så en kryptoanalytiker har mulighed for at aflytte kommunikationen under denne passage. Dette problem kan vi ikke umiddelbart løse.

Kørslen af nøglegenererings samt ind- og afkodningsprogrammerne kan vi heller ikke gøre sikker. Vi kan sørge for at de kører på den mest beskyttede måde operativsystemet tillader, men vi har ingen mulighed for at få dem kørt i speciel sikker hardware. De kritiske krypteringsrutiner bør eventuelt gøres til helt selvstændige programmer, der kører uafhængigt af resten af systemet, for at sikre dem mest muligt. Det kan f.eks gøres ved at de foretages i krypteringsboks.

### 12.1.4 Grænseflade - Opsummering

Det elektroniske postsystem får med kryptering 4 ekstra ordrer.

1. Generer nøgle (til mig), indsæt den offentlige nøgle i den globale nøgletabel og indkod den hemmelige nøgle under et password, som (jeg) indtaster.

2. Indkod og send meddelelse ved at bede om modtagerens brugernummer.
3. Afkod med password og vis meddelelse.
4. Administrer nøgletabel, og herunder :
   (a) Indsæt en ny bruger i tabellen.
   (b) Slet en bruger fra tabellen.
   (c) Ret i en bruger i tabellen.

   I denne tabel opbevares evt. brugerprofiler også.

### 12.1.5 Pseudokode

De følgende 3 algoritmer giver pseudokoden for de første 3 af de 4 krypterings-funktioner der skal ligge i det elektroniske post-system.

**Algoritme 12.1 Opret nøgler**

1. Prompt for password.
2. Beregn Sæd := $F(password, \text{tid})$ [1]
3. Skafnøgler(Sæd, Brugerprofil.blokstørrelse)
4. Indsæt offentlig-nøgle i offentlig tabel.
5. Giv den hemmelige nøgle $d$ til brugeren.
6. Lad DES := Desnøgle(password)
7. Gem d_krypt := Deskrypt(d).
8. Gem des_krypt := $\mathcal{E}$(DES), som verifikationsnøgle.

$F$ er en eller anden funktion, der giver sæd til tilfældighedsgenerator på baggrund af tiden og $P$, den skal have så stor værdimængde som muligt. $\mathcal{E}$ er en envejsfunktion.

**Algoritme 12.2 Afkod**

[1] Tiden skal med for at det samme password ikke giver samme nøgler.

1. Læs Ciffertekst $c$.
2. Prompt for password
3. Lad DES := Desnøgle(password)
4. Hvis $\mathcal{E}$(DES) $\neq$ des_krypt så STOP
5. Lad $d$ :=des_dekrypt(d_krypt)
6. Lad $m$ := RSA_Verificer(RSA_afkod(c, d), e)

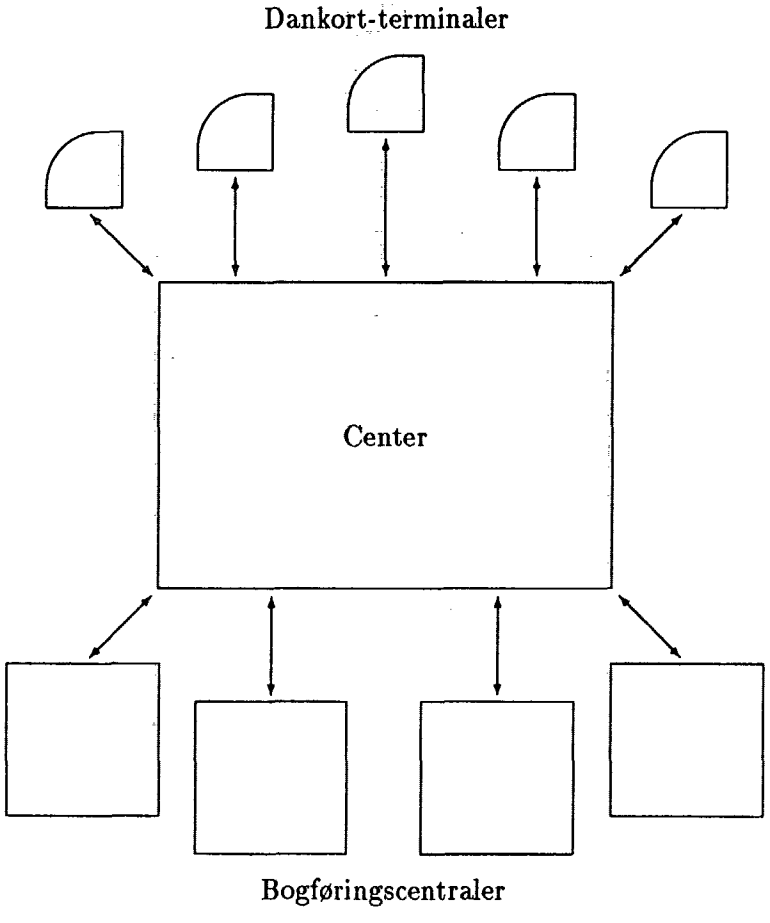
**Algoritme 12.3 Indkod**

1. Læs Klartekst $m$.
2. Prompt for password
3. Lad DES := Desnøgle(password)
4. Hvis $\mathcal{E}$(DES) $\neq$ des_krypt så STOP
5. Lad $d$ :=des_dekrypt(d_krypt)
6. Søg modtagers nøgle $e$ i offentlig tabel.
7. Hvis den ikke findes der, så led i personlig tabel.
8. Hvis fundet så lad $c$ := RSA_indkod(RSA_signer(m,d),e)
9. Send(c)

## 12.2 RSA i DANKORT-netværket

For at fuldende historien om RSA, vil vi i dette afsnit give en kort beskrivelse af en eksisterende RSA-anvendelse, nemlig DANKORT netværket.

Siden først i 80'erne har det været muligt at udføre elektroniske betalingstransaktioner mellem butikker og pengeautomater til pengeinstitutterne, ved brug af det såkaldte "Dankort".




**Figur 12.1:** Dankortnetværkets opbygning

### 12.2.1 Opbygningen af Dankortnettet

Systemet er opbygget som et stjernenet fig 12.1, hvor de enkelte enheder via det offentlige datanet overfører transaktionerne til et center. Herfra distribueres de en gang i døgnet via pengeinstitutterne til bogføringscentralerne, hvor de egentlige posteringer på kortholdernes konti finder sted.

Et sådant system kan kun fungere, hvis man med meget høj sikkerhed kan verificere kortholderens autenticitet og transaktionens integritet. Verifikation af kortholderens identitet er baseret på en firecifret personlig kode, PIN-koden, som tildeles kortholderen når Dankortet oprettes, og herefter kun eksisterer i kodet form i det centrale Dankortregister, hvor også selve verifikationen sker.

Kryptograferingen af kommunikationen mellem de enkelte enheder og centralen foregår med hemmelignøglesystemet DES, mens udvekslingen af hemmelige nøgler er baseret på RSA.

### 12.2.2 Dankort protokollen

#### Udveksling af hemmelige nøgler

For at sikre fortrolig kommunikation mellem de enkelte Dankort terminaler og Dankortcentralen, skal der til enhver tid eksistere et sæt hemmelige nøgler for hver enhed, der kun kendes af centralen og enheden selv. For at opnå en høj sikkerhed, skal disse nøgler udskiftes med jævne mellemrum. Dette kunne gøres ud fra en tilfældighedsfunktion, men sker i praksis, hver gang der detekteres en datafejl på kommunikationslinien. Dette sker typisk 4-8 gange om dagen. Når et nyt sæt nøgler skal genereres er protokollen følgende :

1. Terminalen genererer et tilfældigt tal, Session key.
2. Session key indkodes med terminalens offentlige RSA-nøgle og transmitteres.
3. Centralen afkoder Session key med den hemmelige RSA-nøgle.
4. Centralen opretter de nødvendige nøgler til hemmelignøglesystemet : DES-nøglerne.

5. DES-nøglerne indkodes med et hemmelignøglesystem under Session key og sendes til den pågældende terminal.

6. Terminalen, der kender Session key, afkoder DES-nøglerne, som hermed findes i begge ender af linien, uden nogensinde at have være transmitteret i klartekst.

### Kryptografering af transaktioner

Når en kortholder ønsker at udføre en transaktion foregår identifikation og transmission i store træk på følgende måde (al kommunikation er her krypteret med DES) :

1. Magnetkortet indsættes i kortlæseren og PIN koden indtastes.

2. Terminalen udfører en eenvejsfunktion på PIN koden XOR'et med indholdet af magnetstriben. Resultatet sendes til centralen.

3. Dankortcentralen sammenholder de modtagne oplysninger med en tabel over brugerne, hvor kun resultaterne af eenvejsfunktionerne er lagret.

4. Hvis identifikationen er rigtig meddeles dette til pengeautomaten og transaktionen kan begynde.

## 12.2.3 Implementering

Systemet opererer med en blokstørrelse og dermed en modulus på 512 bits, hvilket svarer til 154 decimale cifre.

### Indkodning

For at gøre de enkelte terminaler så enkle og prisbillige som muligt, har man modificeret RSA protokollen lidt, således at den offentlige nøgle vælges meget lille (typisk en decimal). Dette betyder at indkodningen bliver beregningsmæssigt let og det forringer ikke sikkerheden, da indkodningsnøglen er offentligt kendt. Man skal dog overveje at for meddelelser $m$ mindre end $\sqrt[e]{n}$ kan klarteksten dekrypteres ved at beregne $m = \sqrt[e]{m^e}$. Antallet af disse "dårlige meddelelser" er dog så forsvindende lille så en kryptoanalytiker ikke ved *hvilke* ciffertekster, der kan dekrypteres.

**afkodning**

De "tunge" beregninger er således begrænset til Dankortcentralen, som da også er udstyret med et stykke hardware, FAP (Fast Arithmetic Processor)[32], der er designet til at behandle meget store tal med stor hastighed. Med FAP installeret er det muligt at transformere een blok på ca 0,1 sek. Skal dette gøres uden FAP, vil det på samme system tage op mod 1 minut.

Dankort systemet benytter desuden en optimering af afkodningsalgoritmen, der sænker køretiden med en faktor 4. Optimeringen benytter at kendskabet til den hemmelige nøgle også tillader kendskab til faktorerne $p$ og $q$. Afkodningen foregår da ved at afkode ciffertekst en med hensyn til $p$ og $q$ hver for sig:

$$\begin{aligned} m_1 &= c^d \text{ MOD } p \\ m_2 &= c^d \text{ MOD } q \end{aligned}$$

Dette betyder at

$$\begin{aligned} m &\equiv m_1 \bmod p \\ m &\equiv m_2 \bmod q \end{aligned}$$

som ved hjælp af den kinesiske restsætning giver løsningen

$$m \equiv (m_1 \cdot p^{q-1} + m_2 \cdot q^{p-1}) \bmod n \tag{12.1}$$

Størrelserne $p^{q-1}$ og $q^{p-1}$ kan beregnes en gang for alle ved selve nøgleoprettelsen

$$\begin{aligned} k_1 &= p^{q-1} \text{ MOD } n \\ k_2 &= q^{p-1} \text{ MOD } n \end{aligned}$$

så ligning 12.1 giver anledning til følgende afkodnings formel:

$$m = (k_1 \cdot m_1 + k_2 \cdot m_2) \text{ MOD } n \tag{12.2}$$

Med denne optimering har man i Dankort systemet opnået en afkodningstid (i software) på $60/4 = 17$ sekunder.

### Nøgleoprettelse

Dankort systemet genererer primtallene som stærke primtal (jævnfør kapitel 9) og tester dem med Miller-Rabins primtalstest (se kapitel 4). Til at finde inverse i forbindelse med nøglegenerering benyttes Euclids modificerede algoritme (se kapitel 11).

Oprettelse af et nyt sæt RSA-nøgler varer 15-30 minutter, hvis de genereres i software alene. Med FAP installeret er beregningstiden 30 - 60 sek.

### Nøgleadministration

For at sikre de hemmelige nøgler bedst muligt benytter Dankortnettet de såkaldte **krypteringsbokse**. Dette er en selvstændig datamat, der både indeholder nøgler og de algoritmer, der benytter nøglerne. Denne datamat er indkapslet i en metalboks, der er indrettet således at ethvert forsøg på fysisk indtrængen vil resultere i at krypteringsboksen sletter alle sine data. Det kan være et forsøg på at skrue de enkelte skruer ud, eller et forsøg på at ødelægge den omgivende metalplade. I dette tilfælde, vil en tynd printplade, under det armerede metal sørge for sletningen af alle data. Boksen er ydermere sikret således, at adgangen til operativsystemet er blokeret. Således er en krypteringsboks i den bogstavligste forstand en "blackbox", som udfører ind- og afkodningsfunktionerne uden på noget tidspunkt at afsløre de involverede nøgler.

Adgang til brug af en sådan krypteringsboks er sikret med sit eget hierarki af passwords og beskyttelse af den bygning i hvilken, den befinder sig. Hvis man endelig kommer igennem hierarkiet af passwords, er boksens eneste funktion, at indkode en meddelelse og afkode den igen.

# Kapitel 13

# Konklusion

I problemformuleringen stillede vi tre spørgsmål om RSA-kryptosystemet, som vi ønskede at få besvaret gennem dette projekt. Del 1 af rapportens giver baggrundsmateriale, mens resten af rapportensøger at besvare spørgsmålene.

Kapitlerne om edb-sikkerhed og kryptologi, beskrev det primære anvendelsesområde for kryptografi og specielt offentlig-nøgle systemer, som sikring af fortrolighed og integritet i netværkskommunikation. Vi beskrev også i kapitlet om anvendelser af offentlig-nøgle kryptografi, at offentlig-nøgle systemer kunne have andre, mere utraditionelle anvendelser, såsom sikring af autenticitet og pokerspil.

Kapitlerne om RSA-kryptosystemet og RSA-sikkerhed beskrev RSA-systemet og viste at systemet, udfra de sidste 13 års analyse, må siges at være et sikkert system *idag*.

De sidste tre kapitler i rapporten beskrev den praktiske anvendelse af RSA-systemet. Vores implementering og andres erfaringer viser klart at RSA ikke er realistisk til praktiske anvendelser uden speciel hardware.

Tilsammen giver dette et billede af RSA-kryptosystemet, som et sikkert og meget bredt anvendeligt kryptosystem. Med til billedet hører imidlertid også at de forholdsvis lave indkodningshastigheder, kan blive en hæmsko for udbredelsen af systemet. RSA-systemet er – og vil formentlig altid være meget langsommere end hemmelig-nøgle systemer som f.eks. DES-systemet. Systemet har imidlertid *en* sikker anvendelse. Nøgleudveksling i hybridsystemer, som f.eks. DANKORT-systemet. Her er de lave indkodningshastigheder ikke noget problem, og man får løst problemerne med nøgleudveksling fuldstændigt.

Et interessant aspekt ved at bygge kryptografiske systemer på matematiske problemer, er at en eventuel løsning af et sådant problem vil få vidtrækkende konsekvenser, og formentlig ikke vil blive modtaget med begejstring. Hvis man for eksempel finder en meget hurtig måde at faktorisere på, vil mange institutioners edb-systemers sikkerhed være kompromitteret. Derfor vil man måske slet ikke være interesseret i at offentliggøre en sådan hurtig algoritme.

# Appendiks A

# Procedure Div

```
 1
 2 !-------------------------------------------------------------------------+
 3 |Navn     : D_Div                                                         |
 4 |                                                                         |
 5 |Funktion : d_div dividerer this heltal med h. This heltal destrueres     |
 6 |           undervejs.                                                    |
 7 |                                                                         |
 8 |           Først skilles alle specialtilfældene fra, efter nedstående    |
 9 |           tabel :                                                       |
10 |                                                                         |
11 |           This       h        resultat                                 |
12 |           ------------------------------------------------------------ |
13 |             -        +        -(-this heltal).div(h)                   |
14 |             +        -        - this heltal.div(-h)                    |
15 |             -        -        (-this heltal).div(-h)                   |
16 |                      0        Fejl                                     |
17 |           This   =   h        kvotient = 1, rest = 0                   |
18 |           This   <   h        kvotient = 0, rest = this heltal         |
19 |                  h < grundtal this heltal.D_enkeltdiv(h.ciffer)         |
20 |               ellers          Normal                                   |
21 |                                                                         |
22 |           Bemærk at h IKKE må ødelægges af en D_div. Bemærk også at     |
23 |           alle negeringer i de tre tilfælde med negativt fortegn fore-|
24 |           går i denne procedure, så der ikke skal oprettes nye heltal.|
25 |                                                                         |
26 |           Normaltilfældet efter algoritmen beskrevet i rapporten.       |
27 |                                                                         |
```

```
28 |Inddata  : This heltal - Det heltal der er dividend. This heltal       |
29 |                       - destrueres undervejs.                         |
30 |           h           - Det heltal der er divisor.                    |
31 |                                                                       |
32 |Uddata   : D_div       - et heltal der er kvotienten.                  |
33 |           Rest        - Et heltal der er resten.                      |
34 |                                                                       |
35 +-----------------------------------------------------------------------;
36
37 ref(Heltal) Procedure D_Div(h, Rest); name Rest; ref(Heltal) h, Rest;
38 Begin
39   Procedurekald := Procedurekald + 1;
40
41   if Negativ and h.Positiv then
42   Begin
43     ref(Heltal) Kvotient;
44
45     Negativ          := NOT Negativ;
46     Kvotient         :- This Heltal.D_Div(h, Rest);
47     if NOT Rest.Ligmed(NUL) then Rest :- h.Sub(Rest);
48     Kvotient.Negativ := NOT Kvotient.Negativ;
49     D_Div            :- Kvotient;
50   End else
51
52   if Positiv and h.Negativ then
53   Begin
54     ref(Heltal) Kvotient;
55     h.Negativ        := NOT h.negativ;
56     Kvotient         :- This Heltal.D_Div(h, Rest);
57     h.Negativ        := NOT h.negativ;
58     Kvotient.Negativ := NOT Kvotient.Negativ;
59     if NOT Rest.Ligmed(NUL) then Rest :- Rest.Add(h);
60     D_Div            :- Kvotient;
61   End else
62
63   if Negativ and h.Negativ then
64   Begin
65     h.Negativ := NOT h.negativ;
66     Negativ   := NOT negativ;
67     D_Div     :- This Heltal.D_Div(h, Rest);
68     h.Negativ := NOT h.Negativ;
69     if NOT Rest.Ligmed(NUL) then Rest.Negativ := NOT Rest.Negativ;
```

```
 70   End else
 71
 72   if h.Ligmed(NUL) then Fejl("Division med NUL !",10) else
 73
 74   if Ligmed(h) then
 75   Begin
 76     Rest   :- New Heltal(grundtal); ! -- Nul --;
 77     D_Div  :- NUL.EnkeltAdd(1);     ! -- Een --;
 78   End else
 79
 80   if mindre_end(h) then
 81   Begin
 82     Rest   :- This Heltal;
 83     D_Div  :- New Heltal(grundtal); ! -- Nul --;
 84   End else
 85
 86   if h.cardinal < 2 then
 87   Begin
 88     D_Div :- D_Enkeltdiv(h.first qua ciffer.tal, Rest);
 89   End else
 90
 91   Begin
 92     Boolean norm;
 93     integer normfaktor, q, to_første_cifre, D_cifre;
 94     ref(Ciffer) C1, C2, C3, D1, D2;
 95     ref(Heltal) Dividend, Divisor, Kvotient, Remainder, Stak;
 96
 97     Dividend  :- This Heltal;
 98     Divisor   :- h.kopi;
 99     Kvotient  :- New Heltal (grundtal);
100     Remainder :- New Heltal (grundtal);
101     D_cifre   := Dividend.Cardinal;
102     D1        :- Divisor.First;
103
104     if D1.tal < grundtal // 2 then
105     Begin
106       norm := TRUE;
107       normfaktor := grundtal // (D1.tal + 1);
108       Dividend :- Dividend.D_Enkeltmult(normfaktor);
109       Divisor  :- Divisor.D_Enkeltmult(normfaktor);
110     End;
111
```

```
112     !-- Herefter er første tal i divisor > grundtal//2 --;
113
114     if D_cifre = Dividend.Cardinal then new Ciffer(0).Follow(Dividend);
115
116     ! -- Der SKAL et ekstra ciffer ind forrest i dividenden --;
117
118     D1 :- Divisor.First;
119     D2 :- D1.suc;
120
121     !---- Her nedbrydes dividenden ------;
122
123     if divisor.Cardinal > Dividend.Cardinal // 2 then
124     Begin
125       Stak :- New Heltal(grundtal);
126       while Dividend.Cardinal > Divisor.Cardinal do
127         Dividend.Last.Into(Stak);
128     end else
129
130     Begin
131       Stak     :- Dividend;
132       Dividend :- New Heltal (grundtal);
133       While Dividend.Cardinal < Divisor.Cardinal do
134       Stak.First.Into(Dividend)
135     End;
136
137     ! --- Herefter har dividend længde = divisor. Stak indeholder resten;
138
139     D_cifre := Divisor.Cardinal + 1;
140
141     While Stak.First =/= NONE do
142     Begin
143       Stak.First.Into(Dividend);
144
145       !-- Dividend har et ciffer mere end divisor --;
146
147       C1 :- Dividend.First;
148       C2 :- C1.suc;
149       C3 :- C2.suc;
150
151       to_første_cifre := C1.tal * grundtal + C2.tal;
152       q  := if C1.tal = D1.tal then grundtal - 1
153                                else to_første_cifre // D1.tal;
```

```
154
155       While (D2.tal * q - C3.tal) / grundtal > to_første_cifre-D1.tal*q do
156         q:= q - 1;
157
158       Dividend :- Dividend.Normaliser; !- ellers virker Mindreend IKKE -;
159       Dividend :- Dividend.D_Sub(Divisor.enkeltmult(q));
160
161       While Dividend.Negativ do
162       Begin
163         Dividend :- Dividend.D_Add(Divisor);
164         q        := q - 1;
165       end;
166
167       if q >= grundtal then fejl("Kvotient for stor i div",1);
168
169       While D_cifre > Dividend.Cardinal do
170         New Ciffer(0).Follow(Dividend);
171
172       Dividend.First.Into(Remainder);
173       New Ciffer(q).Into(Kvotient);
174     End *** while Mere Stak ***;
175
176     While NOT Dividend.Empty do Dividend.First.Into(Remainder);
177     Begin
178       Ref(Heltal) R; !--Dummy rest, som kastes bort --;
179
180       if norm then Remainder :- Remainder.Enkeltdiv(normfaktor,R);
181     End;
182
183     Rest   :- Remainder.Normaliser;
184     D_Div  :- Kvotient.Normaliser;
185
186   End ** tallene er positive og dividend > divisor **;
187 End * D_Div *;
188
```

# Appendiks B

# Proces

## B.1 Forløbsbeskrivelse

Projektet blev startet op i Februar 89, med udgangspunkt i et emne der hed *"koder"* : Kryptografi og fejlkorrigerende/fejldetekterende koder. Formålet var fra starten at kombinere fagområderne matematik og datalogi i et projekt, der skulle forløbe over et år.

Forårssemesteret gik med at tilvejebringe den nødvendige faglige baggrund, specielt indenfor talteori, og med at indkredse emnet. Dette ændredes hurtigt fra fejlkorrigerende koder til forskellige kryptografiske metoder. Emnet indsnævredes derefter yderligere til offentlig nøglesystemer, som endelig udkrystalliseredes i RSA-kryptosystemet. Vi var således rundt om meget andet, end det der står skrevet i rapporten. Det drejer sig for eksempel om kommunikationsteori, herunder redundans, NP-fuldstændighed, diskrete logaritmer, og to andre offentlignøgle systemer ved navn Knapsack og Lu-Lee's kryptosystem. Rapportens 5 første kapitler blev også skrevet i en første udgave i denne periode.

I efteråret har vi koncentreret os om at benytte materialet fra forårets arbejde til at beskrive RSA-systemet, og om at designe og implementere RSA-værktøjet.

## B.2 Forløbsvurdering

Processen har forløbet meget ideelt for os, da der ikke har stået nogle hindringer af betydning i vejen for vores aktiviteter:

Det var meget oplagt hvorledes, vi skulle indkredse emnet, og det har hele tiden været tydeligt at vi fulgte et "godt" spor.

Det nødvendige baggrundsmateriale afgrænsedes på en naturlig måde gennem de referencer, litteraturen om RSA angav. Desuden kunne vi på et meget tidligt tidspunkt opstille nogle prototyper, i form af nogle C-programmer, der demonstrerede principperne for RSA og muliggjorde beregning af de faktiske RSA eksempler.

Vi har to gange i forløbet kontaktet ophavsmanden Ronald Rivest skriftligt og i begge tilfælde fået meget positive og værdifulde svar - både fra ham, og hans kolleger. H.B Hansen fra datalogiuddannelsen på RUC har flere gange været parat med gode råd og ideer og Finn Munch fremskaffede meget tidligt en artikel, der satte os på rette spor. Endvidere har folkene omkring dankortprojektet været meget behjælpelige med oplysninger om den praktiske implementering. Endelig har Michael Pedersen ydet en anseelig indsats med hensyn til igen og igen at korrekturlæse de matematisk følsomme afsnit og iøvrigt fremskaffe værdifuldt materiale om talteori. Vi har altså kunnet trække på mange forskellige menneskers viden og på den måde fået en god fremdrift i vores læreproces.

Alt i alt er vi alle meget tilfredse med den måde processen har forløbet på og alle har vi såvel fagligt og erfaringsmæssigt fået et stort udbytte af det forløbne år.

# Appendiks C

# Litteraturliste

Den følgende litteraturliste indeholder de bøger og artikler, der er refereret til igennem projektet. Der er således ikke tale om en liste, der omfatter al litteratur om emnet. Dog mener vi at det meste af den litteratur, der findes om RSA-kryptosystemet er repræsenteret.

Hvis man ønsker at læse mere om kryptologi og datasikkerhed, kan vi anbefale [16] som omhandler begge dele, og [5] der kun handler om kryptologi, og som begge er godt skrevet. [31] indeholder en del om kryptografi, primtalstests og faktorisering. Vi har brugt den en del, ikke fordi den er god, men fordi den indeholdt det stof vi skulle bruge. [33] indeholder et helt kapitel kun om primtalstests, og har stort set alle vigtige tests med. Den er imidlertid ret svært tilgængelig. Hvis man vil vide mere om implementation af talteoretiske funktioner er [30] biblen. [48] handler specielt om implementering af RSA-systemet. Er man historisk interresseret i kryptografi, er [29] og [25] oplagte.

# Litteratur


[1] Tom M. Apostol : *Introduction to Analytic Number Theory.* Springer-Verlag, 1976.

[2] Bob Blakley , G. R. Blakley : *Security of Number Theoretic Public Key Systems Against Random Attack I-II-III.* Cryptologia Vol 2, nr 4, side 305-321, 1978. Vol 3, nr 1, side 29-42, 1979. Vol 3, nr 2, side 105-118, 1979.

[3] G. R. Blakley, I. Borosh : *Rivest-Shamir-Adleman Public Key Cryptosystems do not always conceal messages.* Computers and Mathematics with Applications nr 5 1979, side 169-178.

[4] Gunnar Bomann : *Matematik, Gads fagleksikon.* Gad København, 1979

[5] Giles Brassard : *Modern cryptology.* Lecture Notes in Computer Science 325. Springer Verlag, Berlin, Heidelberg, 1988.

[6] Ernest F. Brickell, Andrew M. Odlyzko : *Cryptanalysis : A Survey of Recent Results.* Fra [26], side 578-593.

[7] Benny Chor og Uded Goldreich : *RSA/Rabin Bits are* $\frac{1}{2} + \frac{1}{Poly(\log n)}$ *secure.* Fra [9]

[8] Per Christoffersson, Stig-Arne Ekhall, Vivike Fåk (ed), Siegfried Herda, Pakka Mattila, Wyn Price, Kjell-Ove Widman : *Crypto Users Handbook. A guide for implementors of cryptographic protection in computer systems.* North Holland, 1988.

[9] G. R. Blakley, David Chaum (ed) : *Advances in Cryptology. Procedings of Crypto 84.* Lecture Notes in Computer Science 196. Springer Verlag, Berlin, Heidelberg, 1985.

[10] Hugh C. Williams (ed) : *Advances in Cryptology. Procedings of Crypto 85.* Lecture Notes in Computer Science 218. Springer Verlag, Berlin, Heidelberg, 1985.

[11] Dansk Standardiseringsråd : *Kryptografi og datasikkerhed.* Dansk standardiseringsråd, Hellerup, 1988.

[12] J. A. Davis, D. B. Holdridge, G. J. Simmons : *Status Report on Factoring (at the Sandia National Labs).* Fra [18] side 183-215.

[13] J. A. Davis, D. B. Holdridge, G. J. Simmons : *Factorization of Large integers on a Massively Parallel Computer.* Fra [19] side 235-244.

[14] Martin D. Davis og Elaine J. Weyukner : *Computability, Complexity and Languages.* Academic Press, 1983.

[15] John M. DeLaurentis : *A Further Weakness in the common modulus protocol for the RSA Crypto-algorithm.* Cryptologia, Vol. 8, nr 3, 1984.

[16] Dorothy Elizabeth Robling Denning : *Cryptography and Data Security.* Addison-Wesley Publishing Company, 1983.

[17] Whitfield Diffie og Martin E. Hellman : *New Directions in Cryptography.* Oprindelig bragt i IEEE Transactions on information Theory, Vol 22, Nr 6, November 1976. Taget fra [51].

[18] T. Beth, N. Cot, T. Ingemarsson (ed) : *Advances in Cryptology. Procedings of Eurocrypt 84. A workshop on the Theory and Applications of Cryptographic Techniques. Paris, France, April 1984.* Lecture Notes in Computer Science 209. Springer Verlag, Berlin, Heidelberg, 1985.

[19] Cristoph G. Günther (ed) : *Advances in Cryptology. Procedings of Eurocrypt 88. A workshop on the Theory and Applications of Cryptographic Techniques. Davos, Switzerland, May, 1988.* Lecture Notes in Computer Science 330. Springer Verlag, Berlin, Heidelberg, 1988.

[20] Lars Frank : *EDB-sikkerhed.* Samfundslitteratur, 1987.

[21] Anthony M. Gaglione : *Some Complexity Theory for Cryptography.* Tidsskift, 1987.

[22] B. A. Gordon : *Strong Primes are Easy to Find.* Fra [18] side 216-223.

[23] Raouf N. Gorgui-Naguib, Satnam s. Dlay : *Properties of the Euler Totient Function Modulo 24 and some of its Cryptographic Implications.* Fra [19] side 267-274.

[24] Tore Herlestam : *Critical Remarks on Some Public-key Cryptosystems.* Bit 18, 1978, side 403-496.

[25] Andrew Hodges : *Alan Turing. The Enigma.* Burnett Books Ltd, 1983.

[26] Gustavus J. Simmons (Ed.) : *Proceedings of the IEEE may 1988. Cryptography.* Institute of Electrical and Electronics Engineers, Inc. 1988.

[27] Ben Johnsen : *Kryptografi, primtall og Riemanns hypotese.* Normat nr. 1, 1986.

[28] Wiedren de Jonge, David Chaum : *Attacks on Some RSA-signatures.* Fra [10].

[29] David Kahn : *The Codebreakers. The story of secret writing.* MacMillan Publishing Company, Inc. New york, 1967.

[30] Donald E. Knuth : *The art of Computer Programming. Volume 2 : Seminumerical Algorithms. 2. ed.* Addison Wesley Publishing Company, 1982.

[31] Neal Koblitz : *A Course in Number Theory and Cryptography.* Springer Verlag, 1987.

[32] Koschanski: *Developing an RSA chip.* Fra crypto 1984 eller 1988. Side 350-357.

[33] Evangelos Kranakis : *Primality and Cryptography.* John Wiley & Sons, 1986.

[34] Erik Kristensen og Ole Rindung : *Matematik 2.2.* GAD, København, 1977.

[35] S. Lakshmivarahan : *Algorithms for Public Key Cryptosystems : Theory and application.* Advances in Computer, vol 22, side 45-108. Academic Press, 1983.

[36] Leslie Lamport : LaTeX, *A Document Preparation System.* Addison-Wesley Publishing Company, 1986.

[37] Ralph Merkle : *Protocols for Public-key Cryptosystems.* Fra [51].

[38] Carl H. Meyer og Stephen M. Matyas : *Cryptography : A New Dimension in Computer Data Security.* John Wiley & Sons, 1982.

[39] Jude H. Moore : *Protocol Failures in Cryptosystems.* Fra [26].

[40] *Brevveksling med pengeinstitutternes Betalingsservice.*

[41] Carl Pomerance : *The search for prime numbers.* Scientific American, ukendt nr., side 169-184.

[42] Carl Pomerance : *Recent developments in primality testing.* The Mathematical Intelligencer , Vol. 3, nr 3, side 169-184, 1981.

[43] Carl Pomerance : *The Quadratic Sieve Factoring Algorithm.* Fra [18] side 169-184.

[44] R. L. Rivest, A. Shamir og L. Adleman : *A Method for Obtaining Digital signatures and Public Key Cryptosystems.* Communications of the ACM, vol. 21, nummer 2, side 120-126. 1978.

[45] Ronald L. Rivest : *Remarks on a Proposed Cryptoanalytic Attack on the M.I.T. Public-Key Cryptosystem.* Cyptologia, Vol. 2, nr 1, side 62-65, Januar 1978.

[46] Ronald L. Rivest : *Critical remarks on "Critical Remarks on Some Public-key Cryptosystems" by T. Herlestam.* Bit 19, 1979, side 274-275.

[47] Ronald L. Rivest : *A desription of a Simgle Chip Implementation of the RSA Cipher.* Lambda 1, 1980.

[48] Ronald L. Rivest : *RSA Chips (Past/Present/Future).* Fra [18].

[49] C. P. Schnorr, W. Alexi : *RSA-bits are* $0.5 + \epsilon$ *secure.* Fra [18] side 113-126.

[50] M. R. Schroeder : *Number theory in science and communication.* Springer-Verlag 1984 og 1986.

[51] Gustavus J. Simmons (ed) : *Secure Communications and Asymmetric Cryptosystems.* AAAS Selected Symposium 69, 1982.

[52] Gustavus J. Simmons : *A Weak Privacy Protocol using the RSA Crypto Algorithm.* Cryptologia. Vol. 7, nr 2, 1983.

[53] Gustavus J. Simmons, M. J. Norris : *Preliminary Comments on the M.I.T. Public-key Cryptosystem.* Cryptologia 1, side 406-414, 1977.

[54] Andrew S. Tannembaum : *Structured computer organisation.* Prentice Hall International, 1984.

[55] Andrew S. Tannembaum : *Computer Networks.* Prentice Hall International, 1981.

[56] H. C. Williams, B. Schmid : *Some Remarks Concerning the M.I.T. Public Key Cryptosystem.* Bit 1979, side 525-538.

# Appendiks D

# Symbolliste

## D.1 Matematiske symboler

| | |
|---|---|
| $a^{n*}$ | $\overbrace{a * a * \ldots * a}^{n}$<br>Se side 8. |
| $a^{-1*}$ | Inverst element til $a$ mht. $*$.<br>Se side 8. |
| $(G, *)$ | $G$ er en gruppe mht. $*$.<br>Se definition 2.1, side 9. |
| $(R, *, \diamond)$ | $R$ er en ring mht. $*$ og $\diamond$.<br>Se definition 2.2, side 9. |
| $(L, *, \diamond)$ | $L$ er et legeme mht. $*$ og $\diamond$.<br>Se definition 2.3, side 10. |
| ORD$(G)$ | Ordenen af en gruppe $G$.<br>Se side 10. |
| ORD$(a, G)$ | Ordenen af elementet $a$ i gruppen $G$.<br>Se side 11. |
| $b \mid a$ | $b$ går op i $a$.<br>Se definition 2.5, side 12. |
| $a$ MOD $b$ | Principal rest af $a/b$.<br>Se side 13. |

| | |
|---|---|
| $a//b$ | $a$ heltalsdivideret med $b$. |
| $\pi(x)$ | Antallet af primtal mindre end $x$.<br>Se definition 2.6, side 15. |
| $\mathrm{SFD}(u, v)$ | Største fælles divisor mellem $u$ og $v$.<br>Se definition 2.7, side 16. |
| $\mathrm{MFM}(u, v)$ | Mindste fælles multiplum mellem $u$ og $v$.<br>Se definition 2.8, side 17. |
| $\#$ | Antal elementer i en mængde. |
| $\phi(n)$ | Eulers phi-funktion. Antallet af tal, mindre end $n$, der er indbyrdes primiske med $n$.<br>Se definition 2.9, side 20. |
| $u \equiv v \bmod m$ | Kongruens. $u$ MOD $m = v$ MOD $m$.<br>Se side 22. |
| $G_n$ | Restklassegruppen modulus $n$.<br>Se side 23. |
| $\mathbf{GF}(p)$ | Galois legeme.<br>Se side 26. |
| $\left(\frac{a}{p}\right)$ | Legendres symbol. $p$ er et primtal.<br>Se definition 2.12, side 39. |
| $\left(\frac{a}{n}\right)$ | Jacobis symbol. $n$ er ikke nødvendigvis et primtal.<br>Se definition 2.13, side 42. |
| $\mathcal{O}(n)$ | *'Store O'*-notationen.<br>Se definition 3.1, side 44. |
| $\lfloor x \rfloor$ | Største heltal mindre end eller lig $x$ (trunkering). |

## D.2 Kryptografiske symboler

| | |
|---|---|
| $E(\ldots)$ | Indkodningsalgoritme. |
| $D(\ldots)$ | Afkodningsalgoritme. |
| $M$ | Mængden af klartekster. |

| | |
|---|---|
| $C$ | Mængden af ciffertekster. |
| $m$ | En klartekst. |
| $c$ | En ciffertekst.<br>$c_h$ er en indkodet tekst.<br>$c_s$ er en signeret tekst.<br>$c_{sh}$ er en indkodet og signeret tekst. |
| $k$ | En nøgle. |
| $k_E$ | En indkodningsnøgle. |
| $k_D$ | En afkodningsnøgle. |
| $\mathcal{E}$ | En envejsfunktion.<br>Se definition 7.1, side 133. |
| $\mathcal{S}$ | En smæklås-envejsfunktion.<br>Se definition 7.2, side 134. |
| $n, e, d$ | RSA-nøgler. |
| $p, q$ | Primtal til RSA-nøglerne. |
| $r, s, t$ | Primtal brugt til at konstruere stærke primtal. |
| CBK | CiffertekstBlokKobling.<br>Se afsnit 6.4.1, side 115. |
| DES | Data Encryption Standard.<br>Se afsnit 6.2.2, side 106. |
| FAP | Fast Arithmetic Processor.<br>Se afsnit 12.2.3, side 245. |
| XOR | Exclusiv-or. |

# Liste over eksempler

# Liste over figurer

# Liste over tabeller

# Stikordsregister

1/78 "TANKER OM EN PRAKSIS" - et matematikprojekt.
Projektrapport af: Anne Jensen, Lena Lindenskov, Marianne Kesselhahn og Nicolai Lomholt.
Vejleder: Anders Madsen

2/78 "OPTIMERING" - Menneskets forøgede beherskelsesmuligheder af natur og samfund.
Projektrapport af: Tom J. Andersen, Tommy R. Andersen, Gert Krenøe og Peter H. Lassen
Vejleder: Bernhelm Boss.

3/78 "OPGAVESAMLING", breddekursus i fysik.
Af: Lasse Rasmussen, Aage Bonde Kræmmer og Jens Højgaard Jensen.

4/78 "TRE ESSAYS" - om matematikundervisning, matematiklæreruddannelsen og videnskabsrindalismen.
Af: Mogens Niss
Nr. 4 er p.t. udgået.

5/78 "BIBLIOGRAFISK VEJLEDNING til studiet af DEN MODERNE FYSIKS HISTORIE".
Af: Helge Kragh.
Nr. 5 er p.t. udgået.

6/78 "NOGLE ARTIKLER OG DEBATINDLÆG OM - læreruddannelse og undervisning i fysik, og - de naturvidenskabelige fags situation efter studenteroprøret".
Af: Karin Beyer, Jens Højgaard Jensen og Bent C. Jørgensen.

7/78 "MATEMATIKKENS FORHOLD TIL SAMFUNDSØKONOMIEN".
Af: B.V. Gnedenko.
Nr. 7 er udgået.

8/78 "DYNAMIK OG DIAGRAMMER". Introduktion til energy-bond-graph formalismen.
Af: Peder Voetmann Christiansen.

9/78 "OM PRAKSIS' INDFLYDELSE PÅ MATEMATIKKENS UDVIKLING". - Motiver til Kepler's: "Nova Stereometria Doliorum Vinariom".
Projektrapport af: Lasse Rasmussen.
Vejleder: Anders Madsen.

---

10/79 "TERMODYNAMIK I GYMNASIET".
Projektrapport af: Jan Christensen og Jeanne Mortensen.
Vejledere: Karin Beyer og Peder Voetmann Christiansen.

11/79 "STATISTISKE MATERIALER".
Af: Jørgen Larsen.

12/79 "LINEÆRE DIFFERENTIALLIGNINGER OG DIFFERENTIALLIGNINGSSYSTEMER".
Af: Mogens Brun Heefelt.
Nr. 12 er udgået.

13/79 "CAVENDISH'S FORSØG I GYMNASIET".
Projektrapport af: Gert Kreinøe.
Vejleder: Albert Chr. Paulsen.

14/79 "BOOKS ABOUT MATHEMATICS: History, Philosophy, Education, Models, System Theory, and Works of".
Af: Else Høyrup.
Nr. 14 er p.t. udgået.

15/79 "STRUKTUREL STABILITET OG KATASTROFER i systemer i og udenfor termodynamisk ligevægt".
Specialeopgave af: Leif S. Striegler.
Vejleder: Peder Voetmann Ckristiansen.

16/79 "STATISTIK I KRÆFTFORSKNINGEN".
Projektrapport af: Michael Olsen og Jørn Jensen.
Vejleder: Jørgen Larsen.

17/79 "AT SPØRGE OG AT SVARE i fysikundervisningen".
Af: Albert Christian Paulsen.

18/79 "MATHEMATICS AND THE REAL WORLD", Proceedings af an International Workshop, Roskilde University Centre, Denmark, 1978.
Preprint.
Af: Bernhelm Booss og Mogens Niss (eds.)

19/79 "GEOMETRI, SKOLE OG VIRKELIGHED".
Projektrapport af: Tom J. Andersen, Tommy R. Andersen og Per H.H. Larsen.
Vejleder: Mogens Niss.

20/79 "STATISTISKE MODELLER TIL BESTEMMELSE AF SIKRE DOSER FOR CARCINOGENE STOFFER".
Projektrapport af: Michael Olsen og Jørn Jensen.
Vejleder: Jørgen Larsen

21/79 "KONTROL I GYMNASIET-FORMÅL OG KONSEKVENSER".
Projektrapport af: Crilles Bacher, Per S.Jensen, Preben Jensen og Torben Nysteen.

22/79 "SEMIOTIK OG SYSTEMEGENSKABER (1)".
1-port lineært response og støj i fysikken.
Af: Peder Voetmann Christiansen.

23/79 "ON THE HISTORY AF EARLY WAVE MECHANICS - with special emphasis on the role af realitivity".
Af: Helge Kragh.

---

24/80 a+b "MATEMATIKOPFATTELSER HOS 2.G'ERE".
1. En analyse. 2. Interviewmateriale.
Projektrapport af: Jan Christensen og Knud Lindhardt Rasmussen.
Vejleder: Mogens Niss.

25/80 "EKSAMENSOPGAVER", Dybdemodulet/fysik 1974-79.

26/80 "OM MATEMATISKE MODELLER".
En projektrapport og to artikler.
Af: Jens Højgaard Jensen m.fl.

27/80 "METHODOLOGY AND PHILOSOPHY AF SCIENCE IN PAUL DIRAC's PHYSICS".
Af: Helge Kragh.

28/80 "DIELEKTRISK RELAXATION - et forslag til en ny model bygget på væskernes viscoelastiske egenskaber".
Projektrapport af: Gert Kreinøe.
Vejleder: Niels Boye Olsen.

29/80 "ODIN - undervisningsmateriale til et kursus i differentialligningsmodeller".
Projektrapport af: Tommy R. Andersen, Per H.H. Larsen og Peter H. Lassen.
Vejleder: Mogens Brun Heefelt.

30/80 "FUSIONSENERGIEN - - - ATOMSAMFUNDETS ENDESTATION".
Af: Oluf Danielsen.
Nr. 30 er udgået.

31/80 "VIDENSKABSTEORETISKE PROBLEMER VED UNDERVISNINGSSYSTEMER BASERET PÅ MÆNGDELÆRE".
Projektrapport af: Troels Lange og Jørgen Karrebæk.
Vejleder: Stig Andur Pedersen.
Nr. 31 er p.t. udgået.

32/80 "POLYMERE STOFFERS VISCOELASTISKE EGENSKABER - BELYST VED HJÆLP AF MEKANISKE IMPEDANSMÅLINGER OG MÖSSBAUEREFFEKTMÅLINGER".
Projektrapport af: Crilles Bacher og Preben Jensen.
Vejledere: Niels Boye Olsen og Peder Voetmann Christiansen.

33/80 "KONSTITUERING AF FAG INDEN FOR TEKNISK - NATURVIDENSKABELIGE UDDANNELSER. I-II".
Af: Arne Jakobsen.

34/80 "ENVIRONMENTAL IMPACT AF WIND ENERGY UTILIZATION".
ENERGY SERIES NO. I.
Af: Bent Sørensen
Nr. 34 er udgået.

35/80 "HISTORISKE STUDIER I DEN NYERE ATOMFYSIKS UDVIKLING".
Af: Helge Kragh.

36/80 "HVAD ER MENINGEN MED MATEMATIKUNDERVISNINGEN?".
Fire artikler.
Af: Mogens Niss.

37/80 "RENEWABLE ENERGY AND ENERGY STORAGE".
ENERGY SERIES NO. 2.
Af: Bent Sørensen.

---

38/81 "TIL EN HISTORIETEORI OM NATURERKENDELSE, TEKNOLOGI OG SAMFUND".
Projektrapport af: Erik Gade, Hans Hedal, Henrik Lau og Finn Physant.
Vejledere: Stig Andur Pedersen, Helge Kragh og Ib Thiersen.
Nr. 38 er p.t. udgået.

39/81 "TIL KRITIKKEN AF VÆKSTØKONOMIEN".
Af: Jens Højgaard Jensen.

40/81 "TELEKOMMUNIKATION I DANMARK - oplæg til en teknologivurdering".
Projektrapport af: Arne Jørgensen, Bruno Petersen og Jan Vedde.
Vejleder: Per Nørgaard.

41/81 "PLANNING AND POLICY CONSIDERATIONS RELATED TO THE INTRODUCTION OF RENEWABLE ENERGY SOURCES INTO ENERGY SUPPLY SYSTEMS".
ENERGY SERIES NO. 3.
Af: Bent Sørensen.

42/81 "VIDENSKAB TEORI SAMFUND - En introduktion til materialistiske videnskabsopfattelser".
Af: Helge Kragh og Stig Andur Pedersen.

43/81 1."COMPARATIVE RISK ASSESSMENT OF TOTAL ENERGY SYSTEMS".
2."ADVANTAGES AND DISADVANTAGES OF DECENTRALIZATION".
ENERGY SERIES NO. 4.
Af: Bent Sørensen.

44/81 "HISTORISKE UNDERSØGELSER AF DE EKSPERIMENTELLE FORUDSÆTNINGER FOR RUTHERFORDS ATOMMODEL".
Projektrapport af: Niels Thor Nielsen.
Vejleder: Bent C. Jørgensen.

---

45/82 Er aldrig udkommet.

46/82 "EKSEMPLARISK UNDERVISNING OG FYSISK ERKENDESE-
1+11 ILLUSTRERET VED TO EKSEMPLER".
Projektrapport af: Torben O.Olsen, Lasse Rasmussen og Niels Dreyer Sørensen.
Vejleder: Bent C. Jørgensen.

47/82 "BARSEBÄCK OG DET VÆRST OFFICIELT-TÆNKELIGE UHELD".
ENERGY SERIES NO. 5.
Af: Bent Sørensen.

48/82 "EN UNDERSØGELSE AF MATEMATIKUNDERVISNINGEN PÅ ADGANGSKURSUS TIL KØBENHAVNS TEKNIKUM".
Projektrapport af: Lis Eilertzen, Jørgen Karrebæk, Troels Lange, Preben Nørregaard, Lissi Pedesen, Laust Rishøj, Lill Røn og Isac Showiki.
Vejleder: Mogens Niss.

49/82 "ANALYSE AF MULTISPEKTRALE SATELLITBILLEDER".
Projektrapport af: Preben Nørregaard.
Vejledere: Jørgen Larsen og Rasmus Ole Rasmussen.

50/82 "HERSLEV - MULIGHEDER FOR VEDVARENDE ENERGI I EN LANDSBY".
ENERGY SERIES NO. 6.
Rapport af: Bent Christensen, Bent Hove Jensen, Dennis B. Møller, Bjarne Laursen, Bjarne Lillethorup og Jacob Mørch Pedersen.
Vejleder: Bent Sørensen.

51/82 "HVAD KAN DER GØRES FOR AT AFHJÆLPE PIGERS BLOKERING OVERFOR MATEMATIK ?"
Projektrapport af: Lis Eilertzen, Lissi Pedersen, Lill Røn og Susanne Stender.

52/82 "DESUSPENSION OF SPLITTING ELLIPTIC SYMBOLS".
Af: Bernhelm Booss og Krzysztof Wojciechowski.

53/82 "THE CONSTITUTION OF SUBJECTS IN ENGINEERING EDUCATION".
Af: Arne Jacobsen og Stig Andur Pedersen.

54/82 "FUTURES RESEARCH" - A Philosophical Analysis of Its Subject-Matter and Methods.
Af: Stig Andur Pedersen og Johannes Witt-Hansen.

55/82 "MATEMATISKE MODELLER" - Litteratur på Roskilde Universitetsbibliotek.
En biografi.
Af: Else Høyrup.

Vedr. tekst nr. 55/82 se også tekst nr. 62/83.

56/82 "EN - TO - MANGE" -
En undersøgelse af matematisk økologi.
Projektrapport af: Troels Lange.
Vejleder: Anders Madsen.

---

57/83 "ASPECT EKSPERIMENTET"-
Skjulte variable i kvantemekanikken?
Projektrapport af: Tom Juul Andersen.
Vejleder: Peder Voetmann Christiansen.
Nr. 57 er udgået.

58/83 "MATEMATISKE VANDRINGER" - Modelbetragtninger over spredning af dyr mellem småbiotoper i agerlandet.
Projektrapport af: Per Hammershøj Jensen og Lene Vagn Rasmussen.
Vejleder: Jørgen Larsen.

59/83 "THE METHODOLOGY OF ENERGY PLANNING".
ENERGY SERIES NO. 7.
Af: Bent Sørensen.

60/83 "MATEMATISK MODEKSPERTISE"- et eksempel.
Projektrapport af: Erik O. Gade, Jørgen Karrebæk og Preben Nørregaard.
Vejleder: Anders Madsen.

61/83 "FYSIKS IDEOLOGISKE FUNKTION, SOM ET EKSEMPEL PÅ EN NATURVIDENSKAB - HISTORISK SET".
Projektrapport af: Annette Post Nielsen.
Vejledere: Jens Høyrup, Jens Højgaard Jensen og Jørgen Vogelius.

62/83 "MATEMATISKE MODELLER" - Litteratur på Roskilde Universitetsbibliotek.
En biografi 2. rev. udgave.
Af: Else Høyrup.

63/83 "GREATING ENERGY FUTURES:A SHORT GUIDE TO ENERGY PLANNING".
ENERGY SERIES No. 8.
Af: David Crossley og Bent Sørensen.

64/83 "VON MATEMATIK UND KRIEG".
Af: Berhelm Booss og Jens Høyrup.

65/83 "ANVENDT MATEMATIK - TEORI ELLER PRAKSIS".
Projektrapport af: Per Hedegård Andersen, Kirsten Habekost, Carsten Holst-Jensen, Annelise von Moos, Else Marie Pedersen og Erling Møller Pedersen.
Vejledere: Bernhelm Booss og Klaus Grünbaum.

66/83 "MATEMATISKE MODELLER FOR PERIODISK SELEKTION I ESCHERICHIA COLI".
Projektrapport af: Hanne Lisbet Andersen, Ole Richard Jensen og Klavs Frisdahl.
Vejledere: Jørgen Larsen og Anders Hede Madsen.

67/83 "ELEPSOIDE METODEN - EN NY METODE TIL LINEÆR PROGRAMMERING?"
Projektrapport af: Lone Biilmann og Lars Boye.
Vejleder: Mogens Brun Heefelt.

68/83 "STOKASTISKE MODELLER I POPULATIONSGENETIK"
- til kritikken af teoriladede modeller.
Projektrapport af: Lise Odgård Gade, Susanne Hansen, Michael Hviid og Frank Mølgård Olsen.
Vejleder: Jørgen Larsen.

69/83 "ELEVFORUDSÆTNINGER I FYSIK"
- en test i 1.g med kommentarer.
Af: Albert C. Paulsen.

70/83 "INDLÆRINGS - OG FORMIDLINGSPROBLEMER I MATEMATIK PÅ VOKSENUNDERVISNINGSNIVEAU".
Projektrapport af: Hanne Lisbet Andersen, Torben J. Andreasen, Svend Åge Houmann, Helle Glerup Jensen, Keld Fl. Nielsen, Lene Vagn Rasmussen.
Vejleder: Klaus Grünbaum og Anders Hede Madsen.

71/83 "PIGER OG FYSIK"
- et problem og en udfordring for skolen?
Af: Karin Beyer, Sussanne Blegaa, Birthe Olsen, Jette Reich og Mette Vedelsby.

72/83 "VERDEN IFØLGE PEIRCE" - to metafysiske essays, om og af C.S Peirce.
Af: Peder Voetmann Christiansen.

73/83 ""EN ENERGIANALYSE AF LANDBRUG"
- økologisk contra traditionelt.
ENERGY SERIES NO. 9
Specialeopgave i fysik af: Bent Hove Jensen.
Vejleder: Bent Sørensen.

---

74/84 "MINIATURISERING AF MIKROELEKTRONIK" - om videnskabeliggjort teknologi og nytten af at lære fysik.
Projektrapport af: Bodil Harder og Linda Szkotak Jensen.
Vejledere: Jens Højgaard Jensen og Bent C. Jørgensen.

75/84 "MATEMATIKUNDERVISNINGEN I FREMTIDENS GYMNASIUM"
- Case: Lineær programmering.
Projektrapport af: Morten Blomhøj, Klavs Frisdahl og Frank Mølgaard Olsen.
Vejledere: Mogens Brun Heefelt og Jens Bjørneboe.

76/84 "KERNEKRAFT I DANMARK?" - Et høringssvar indkaldt af miljøministeriet, med kritik af miljøstyrelsens rapporter af 15. marts 1984.
ENERGY SERIES No. 10
Af: Niels Boye Olsen og Bent Sørensen.

77/84 "POLITISKE INDEKS - FUP ELLER FAKTA?"
Opinionsundersøgelser belyst ved statistiske modeller.
Projektrapport af: Svend Åge Houmann, Keld Nielsen og Susanne Stender.
Vejledere: Jørgen Larsen og Jens Bjørneboe.

78/84 "JÆVNSTRØMSLEDNINGSEVNE OG GITTERSTRUKTUT I AMORFT GERMANIUM".
Specialrapport af: Hans Hedal, Frank C. Ludvigsen og Finn C. Physant.
Vejleder: Niels Boye Olsen.

79/84 "MATEMATIK OG ALMENDANNELSE".
Projektrapport af: Henrik Coster, Mikael Wennerberg Johansen, Povl Kattler, Birgitte Lydholm og Morten Overgaard Nielsen.
Vejleder: Bernhelm Booss.

80/84 "KURSUSMATERIALE TIL MATEMATIK B".
Af: Mogens Brun Heefelt.

81/84 "FREKVENSAFHÆNGIG LEDNINGSEVNE I AMORFT GERMANIUM".
Specialerapport af: Jørgen Wind Petersen og Jan Christensen.
Vejleder: Niels Boye Olsen.

82/84 "MATEMATIK - OG FYSIKUNDERVISNINGEN I DET AUTOMATISEREDE SAMFUND".
Rapport fra et seminar afholdt i Hvidovre 25-27 april 1983.
Red.: Jens Højgaard Jensen, Bent C. Jørgensen og Mogens Niss.

83/84 "ON THE QUANTIFICATION OF SECURITY":
PEACE RESEARCH SERIES NO. 1
Af: Bent Sørensen
nr. 83 er p.t. udgået

84/84 "NOGLE ARTIKLER OM MATEMATIK, FYSIK OG ALMENDANNELSE".
Af: Jens Højgaard Jensen, Mogens Niss m. fl.

85/84 "CENTRIFUGALREGULATORER OG MATEMATIK".
Specialerapport af: Per Hedegård Andersen, Carsten Holst-Jensen, Else Marie Pedersen og Erling Møller Pedersen.
Vejleder: Stig Andur Pedersen.

86/84 "SECURITY IMPLICATIONS OF ALTERNATIVE DEFENSE OPTIONS FOR WESTERN EUROPE".
PEACE RESEARCH SERIES NO. 2
Af: Bent Sørensen.

87/84 "A SIMPLE MODEL OF AC HOPPING CONDUCTIVITY IN DISORDERED SOLIDS".
Af: Jeppe C. Dyre.

88/84 "RISE, FALL AND RESURRECTION OF INFINITESIMALS".
Af: Detlef Laugwitz.

89/84 "FJERNVARMEOPTIMERING".
Af: Bjarne Lillethorup og Jacob Mørch Pedersen.

90/84 "ENERGI I 1.G - EN TEORI FOR TILRETTELÆGGELSE".
Af: Albert Chr. Paulsen.

---

91/85 "KVANTETEORI FOR GYMNASIET".
1. Lærervejledning
Projektrapport af: Biger Lundgren, Henning Sten Hansen og John Johansson.
Vejleder: Torsten Meyer.

92/85 "KVANTETEORI FOR GYMNASIET".
2. Materiale
Projektrapport af: Biger Lundgren, Henning Sten Hansen og John Johansson.
Vejleder: Torsten Meyer.

93/85 "THE SEMIOTICS OF QUANTUM - NON - LOCALITY".
Af: Peder Voetmann Christiansen.

94/85 "TREENIGHEDEN BOURBAKI - generalen, matematikeren og ånden".
Projektrapport af: Morten Blomhøj, Klavs Frisdahl og Frank M. Olsen.
Vejleder: Mogens Niss.

95/85 "AN ALTERNATIV DEFENSE PLAN FOR WESTERN EUROPE".
PEACE RESEARCH SERIES NO. 3
Af: Bent Sørensen

96/85 "ASPEKTER VED KRAFTVARMEFORSYNING".
Af: Bjarne Lilletorup.
Vejleder: Bent Sørensen.

97/85 "ON THE PHYSICS OF A.C. HOPPING CONDUCTIVITY".
Af: Jeppe C. Dyre.

98/85 "VALGMULIGHEDER I INFORMATIONSALDEREN".
Af: Bent Sørensen.

99/85 "Der er langt fra Q til R".
Projektrapport af: Niels Jørgensen og Mikael Klintorp.
Vejleder: Stig Andur Pedersen.

100/85 "TALSYSTEMETS OPBYGNING".
Af: Mogens Niss.

101/85 "EXTENDED MOMENTUM THEORY FOR WINDMILLS IN PERTURBATIVE FORM".
Af: Ganesh Sengupta.

102/85 OPSTILLING OG ANALYSE AF MATEMATISKE MODELLER, BELYST VED MODELLER OVER KØERS FODEROPTAGELSE OG - OMSÆTNING".
Projektrapport af: Lis Eilertzen, Kirsten Habekost, Lill Røn og Susanne Stender.
Vejleder: Klaus Grünbaum.

103/85 "ØDSLE KOLDKRIGERE OG VIDENSKABENS LYSE IDEER".
Projektrapport af: Niels Ole Dam og Kurt Jensen.
Vejleder: Bent Sørensen.

104/85 "ANALOGREGNEMASKINEN OG LORENZLIGNINGER".
Af: Jens Jæger.

105/85 "THE FREQUENCY DEPENDENCE OF THE SPECIFIC HEAT AF THE GLASS REANSITION".
Af: Tage Christensen.

"A SIMPLE MODEL AF AC HOPPING CONDUCTIVITY".
Af: Jeppe C. Dyre.
Contributions to the Third International Conference on the Structure of Non - Crystalline Materials held in Grenoble July 1985.

106/85 "QUANTUM THEORY OF EXTENDED PARTICLES".
Af: Bent Sørensen.

107/85 "EN MYG GØR INGEN EPIDEMI",
- flodblindhed som eksempel på matematisk modellering af et epidemiologisk problem.
Projektrapport af: Per Hedegård Andersen, Lars Boye, CarstenHolst Jensen, Else Marie Pedersen og Erling Møller Pedersen.
Vejleder: Jesper Larsen.

108/85 "APPLICATIONS AND MODELLING IN THE MATEMATICS CURRICULUM" - state and trends -
Af: Mogens Niss.

109/85 "COX I STUDIETIDEN" - Cox's regressionsmodel anvendt på studenteroplysninger fra RUC.
Projektrapport af: Mikael Wennerberg Johansen, Poul Katler og Torben J. Andreasen.
Vejleder: Jørgen Larsen.

110/85 "PLANNING FOR SECURITY".
Af: Bent Sørensen

111/85 JORDEN RUNDT PÅ FLADE KORT".
Projektrapport af: Birgit Andresen, Beatriz Quinones og Jimmy Staal.
Vejleder: Mogens Niss.

112/85 "VIDENSKABELIGGØRELSE AF DANSK TEKNOLOGISK INNOVATION FREM TIL 1950 - BELYST VED EKSEMPLER".
Projektrapport af: Erik Odgaard Gade, Hans Hedal, Frank C. Ludvigsen, Annette Post Nielsen og Finn Physant.
Vejleder: Claus Bryld og Bent C. Jørgensen.

113/85 "DESUSPENSION OF SPLITTING ELLIPTIC SYMBOLS II".
Af: Bernhelm Booss og Krzysztof Wojciechowski.

114/85 "ANVENDELSE AF GRAFISKE METODER TIL ANALYSE AF KONTIGENSTABELLER".
Projektrapport af: Lone Biilmann, Ole R. Jensen og Anne-Lise von Moos.
Vejleder: Jørgen Larsen.

115/85 "MATEMATIKKENS UDVIKLING OP TIL RENÆSSANCEN".
Af: Mogens Niss.

116/85 "A PHENOMENOLOGICAL MODEL FOR THE MEYER-NELDEL RULE".
Af: Jeppe C. Dyre.

117/85 "KRAFT & FJERNVARMEOPTIMERING"
Af: Jacob Mørch Pedersen.
Vejleder: Bent Sørensen

118/85 "TILFÆLDIGHEDEN OG NØDVENDIGHEDEN IFØLGE PEIRCE OG FYSIKKEN".
Af: Peder Voetmann Christiansen

---

119/86 "DET ER CANSKE VIST - - EUKLIDS FEMTE POSTULAT KUNNE NOK SKABE RØRE I ANDEDAMMEN".
Af: Iben Maj Christiansen
Vejleder: Mogens Niss.

120/86 "ET ANTAL STATISTISKE STANDARDMODELLER".
Af: Jørgen Larsen

121/86 "SIMULATION I KONTINUERT TID".
Af: Peder Voetmann Christiansen.

122/86 "ON THE MECHANISM OF GLASS IONIC CONDUCTIVITY".
Af: Jeppe C. Dyre.

123/86 "GYMNASIEFYSIKKEN OG DEN STORE VERDEN".
Fysiklærerforeningen, IMFUFA, RUC.

124/86 "OPGAVESAMLING I MATEMATIK".
Samtlige opgaver stillet i tiden 1974-jan. 1986.

125/86 "UVBY,β - systemet - en effektiv fotometrisk spektralklassifikation af B-,A- og F-stjerner".
Projektrapport af: Birger Lundgren.

126/86 "OM UDVIKLINGEN AF DEN SPECIELLE RELATIVITETSTEORI".
Projektrapport af: Lise Odgaard & Linda Szkotak Jensen
Vejledere: Karin Beyer & Stig Andur Pedersen.

127/86 "GALOIS' BIDRAG TIL UDVIKLINGEN AF DEN ABSTRAKTE ALGEBRA".
Projektrapport af: Pernille Sand, Heine Larsen & Lars Frandsen.
Vejleder: Mogens Niss.

128/86 "SMÅKRYB" - om ikke-standard analyse.
Projektrapport af: Niels Jørgensen & Mikael Klintorp.
Vejleder: Jeppe Dyre.

129/86 "PHYSICS IN SOCIETY"
Lecture Notes 1983 (1986)
Af: Bent Sørensen

130/86 "Studies in Wind Power"
Af: Bent Sørensen

131/86 "FYSIK OG SAMFUND" - Et integreret fysik/historie projekt om naturanskuelsens historiske udvikling og dens samfundsmæssige betingethed.
Projektrapport af: Jakob Heckscher, Søren Brønd, Andy Wierød.
Vejledere: Jens Høyrup, Jørgen Vogelius, Jens Højgaard Jensen.

132/86 "FYSIK OG DANNELSE"
Projektrapport af: Søren Brønd, Andy Wierød.
Vejledere: Karin Beyer, Jørgen Vogelius.

133/86 "CHERNOBYL ACCIDENT: ASSESSING THE DATA.
ENERGY SERIES NO. 15.
AF: Bent Sørensen.

---

134/87 "THE D.C. AND THE A.C. ELECTRICAL TRANSPORT IN AsSeTe SYSTE
Authors: M.B.El-Den, N.B.Olsen, Ib Høst Pedersen, Petr Viščor

135/87 "INTUITIONISTISK MATEMATIKS METODER OG ERKENDELSESTEORETISKE FORUDSÆTNINGER"
MASTEMATIKSPECIALE: Claus Larsen
Vejledere: Anton Jensen og Stig Andur Pedersen

136/87 "Mystisk og naturlig filosofi: En skitse af kristendommens første og andet møde med græsk filosofi"
Projektrapport af Frank Colding Ludvigsen
Vejledere: Historie: Ib Thiersen
Fysik: Jens Højgaard Jensen

137/87 "HOPMODELLER FOR ELEKTRISK LEDNING I UORDNEDE FASTE STOFFER" - Resume af licentiatafhandling
Af: Jeppe Dyre
Vejledere: Niels Boye Olsen og Peder Voetmann Christiansen.

138/87 "JOSEPHSON EFFECT AND CIRCLE MAP."

Paper presented at The International Workshop on Teaching Nonlinear Phenomena at Universities and Schools, "Chaos in Education". Balaton, Hungary, 26 April-2 May 1987.

By: Peder Voetmann Christiansen

139/87 "Machbarkeit nichtbeherrschbarer Technik durch Fortschritte in der Erkennbarkeit der Natur"

Af: Bernhelm Booss-Bavnbek
Martin Bohle-Carbonell

140/87 "ON THE TOPOLOGY OF SPACES OF HOLOMORPHIC MAPS"

By: Jens Gravesen

141/87 "RADIOMETERS UDVIKLING AF BLODGASAPPARATUR - ET TEKNOLOGIHISTORISK PROJEKT"

Projektrapport af Finn C. Physant.

Vejleder: Ib Thiersen

142/87 "The Calderón Projektor for Operators With Splitting Elliptic Symbols"

by: Bernhelm Booss-Bavnbek og
Krzysztof P. Wojciechowski

143/87 "Kursusmateriale til Matematik på NAT-BAS"

af: Mogens Brun Heefelt

144/87 "Context and Non-Locality - A Peircan Approach

Paper presented at the Symposium on the Foundations of Modern Physics The Copenhagen Interpretation 60 Years after the Como Lecture. Joensuu, Finland, 6 - 8 august 1987.

by: Peder Voetmann Christiansen

145/87 "AIMS AND SCOPE OF APPLICATIONS AND MODELLING IN MATHEMATICS CURRICULA"

Manuscript of a plenary lecture delivered at ICMTA 3, Kassel, FRG 8.-11.9.1987

By: Mogens Niss

146/87 "BESTEMMELSE AF BULKRESISTIVITETEN I SILICIUM"

- en ny frekvensbaseret målemetode.

Fysikspeciale af Jan Vedde

Vejledere: Niels Boye Olsen & Petr Viščor

147/87 "Rapport om BIS på NAT-BAS"

redigeret af: Mogens Brun Heefelt

148/87 "Naturvidenskabsundervisning med Samfundsperspektiv"

af: Peter Colding-Jørgensen DLH
Albert Chr. Paulsen

149/87 "In-Situ Measurements of the density of amorphous germanium prepared in ultra high vacuum"

by: Petr Viščor

150/87 "Structure and the Existence of the first sharp diffraction peak in amorphous germanium prepared in UHV and measured in-situ"

by: Petr Viščor

151/87 "DYNAMISK PROGRAMMERING"

Matematikprojekt af:
Birgit Andresen, Keld Nielsen og Jimmy Staal

Vejleder: Mogens Niss

152/87 "PSEUDO-DIFFERENTIAL PROJECTIONS AND THE TOPOLOGY OF CERTAIN SPACES OF ELLIPTIC BOUNDARY VALUE PROBLEMS"

by: Bernhelm Booss-Bavnbek
Krzysztof P. Wojciechowski

---

153/88 "HALVLEDERTEKNOLOGIENS UDVIKLING MELLEM MILITÆRE OG CIVILE KRÆFTER"

Et eksempel på humanistisk teknologihistorie
Historiespeciale

Af: Hans Hedal

Vejleder: Ib Thiersen

154/88 "MASTER EQUATION APPROACH TO VISCOUS LIQUIDS AND THE GLASS TRANSITION"

By: Jeppe Dyre

155/88 "A NOTE ON THE ACTION OF THE POISSON SOLUTION OPERATOR TO THE DIRICHLET PROBLEM FOR A FORMALLY SELFADJOINT DIFFERENTIAL OPERATOR"

by: Michael Pedersen

156/88 "THE RANDOM FREE ENERGY BARRIER MODEL FOR AC CONDUCTION IN DISORDERED SOLIDS"

by: Jeppe C. Dyre

157/88 " STABILIZATION OF PARTIAL DIFFERENTIAL EQUATIONS BY FINITE DIMENSIONAL BOUNDARY FEEDBACK CONTROL: A pseudo-differential approach."

by: Michael Pedersen

158/88 "UNIFIED FORMALISM FOR EXCESS CURRENT NOISE IN RANDOM WALK MODELS"

by: Jeppe Dyre

159/88 "STUDIES IN SOLAR ENERGY"

by: Bent Sørensen

160/88 "LOOP GROUPS AND INSTANTONS IN DIMENSION TWO"

by: Jens Gravesen

161/88 "PSEUDO-DIFFERENTIAL PERTURBATIONS AND STABILIZATION OF DISTRIBUTED PARAMETER SYSTEMS:

Dirichlet feedback control problems"

by: Michael Pedersen

162/88 "PIGER & FYSIK - OG MEGET MERE"

AF: Karin Beyer, Sussanne Blegaa, Birthe Olsen,
Jette Reich , Mette Vedelsby

163/88 "EN MATEMATISK MODEL TIL BESTEMMELSE AF PERMEABILITETEN FOR BLOD-NETHINDE-BARRIEREN"

Af: Finn Langberg, Michael Jarden, Lars Frellesen

Vejleder: Jesper Larsen

164/88 "Vurdering af matematisk teknologi
Technology Assessment
Technikfolgenabschätzung"

Af: Bernhelm Booss-Bavnbek, Glen Pate med
Martin Bohle-Carbonell og Jens Højgaard Jensen

165/88 "COMPLEX STRUCTURES IN THE NASH-MOSER CATEGORY"

by: Jens Gravesen

166/88 "Grundbegreber i Sandsynligheds-regningen"

Af: Jørgen Larsen

167a/88 "BASISSTATISTIK 1. Diskrete modeller"

Af: Jørgen Larsen

167b/88 "BASISSTATISTIK 2. Kontinuerte modeller"

Af: Jørgen Larsen

168/88 "OVERFLADEN AF PLANETEN MARS"
Laboratorie-simulering og MARS-analoger undersøgt ved Mössbauerspektroskopi.

Fysikspeciale af:

Birger Lundgren

Vejleder: Jens Martin Knudsen
Fys.Lab./HCØ

169/88 "CHARLES S. PEIRCE: MURSTEN OG MØRTEL TIL EN METAFYSIK."

Fem artikler fra tidsskriftet "The Monist" 1891-93.

Introduktion og oversættelse:

Peder Voetmann Christiansen

170/88 "OPGAVESAMLING I MATEMATIK"

Samtlige opgaver stillet i tiden 1974 - juni 1988

171/88 "The Dirac Equation with Light-Cone Data"

af: Johnny Tom Ottesen

172/88 "FYSIK OG VIRKELIGHED"

Kvantemekanikkens grundlagsproblem i gymnasiet.

Fysikprojekt af:

Erik Lund og Kurt Jensen

Vejledere: Albert Chr. Paulsen og Peder Voetmann Christiansen

---

173/89 "NUMERISKE ALGORITMER"

af: Mogens Brun Heefelt

174/89 " GRAFISK FREMSTILLING AF FRAKTALER OG KAOS"

af: Peder Voetmann Christiansen

175/89 " AN ELEMENTARY ANALYSIS OF THE TIME DEPENDENT SPECTRUM OF THE NON-STATONARY SOLUTION TO THE OPERATOR RICCATI EQUATION

af: Michael Pedersen

176/89 " A MAXIUM ENTROPY ANSATZ FOR NONLINEAR RESPONSE THEORY"

af : Jeppe Dyre

177/89 "HVAD SKAL ADAM STÅ MODEL TIL"

af: Morten Andersen, Ulla Engström, Thomas Gravesen, Nanna Lund, Pia Madsen, Dina Rawat, Peter Torstensen

Vejleder: Mogens Brun Heefelt

178/89 "BIOSYNTESEN AF PENICILLIN - en matematisk model"

af: Ulla Eghave Rasmussen, Hans Oxvang Mortensen, Michael Jarden

vejleder i matematik: Jesper Larsen
biologi: Erling Lauridsen

179a/89 "LÆRERVEJLEDNING M.M. til et eksperimentelt forløb om kaos"

af: Andy Wierød, Søren Brønd og Jimmy Staal

Vejledere: Peder Voetmann Christiansen
Karin Beyer

179b/89 "ELEVHÆFTE: Noter til et eksperimentelt kursus om kaos"

af: Andy Wierød, Søren Brønd og Jimmy Staal

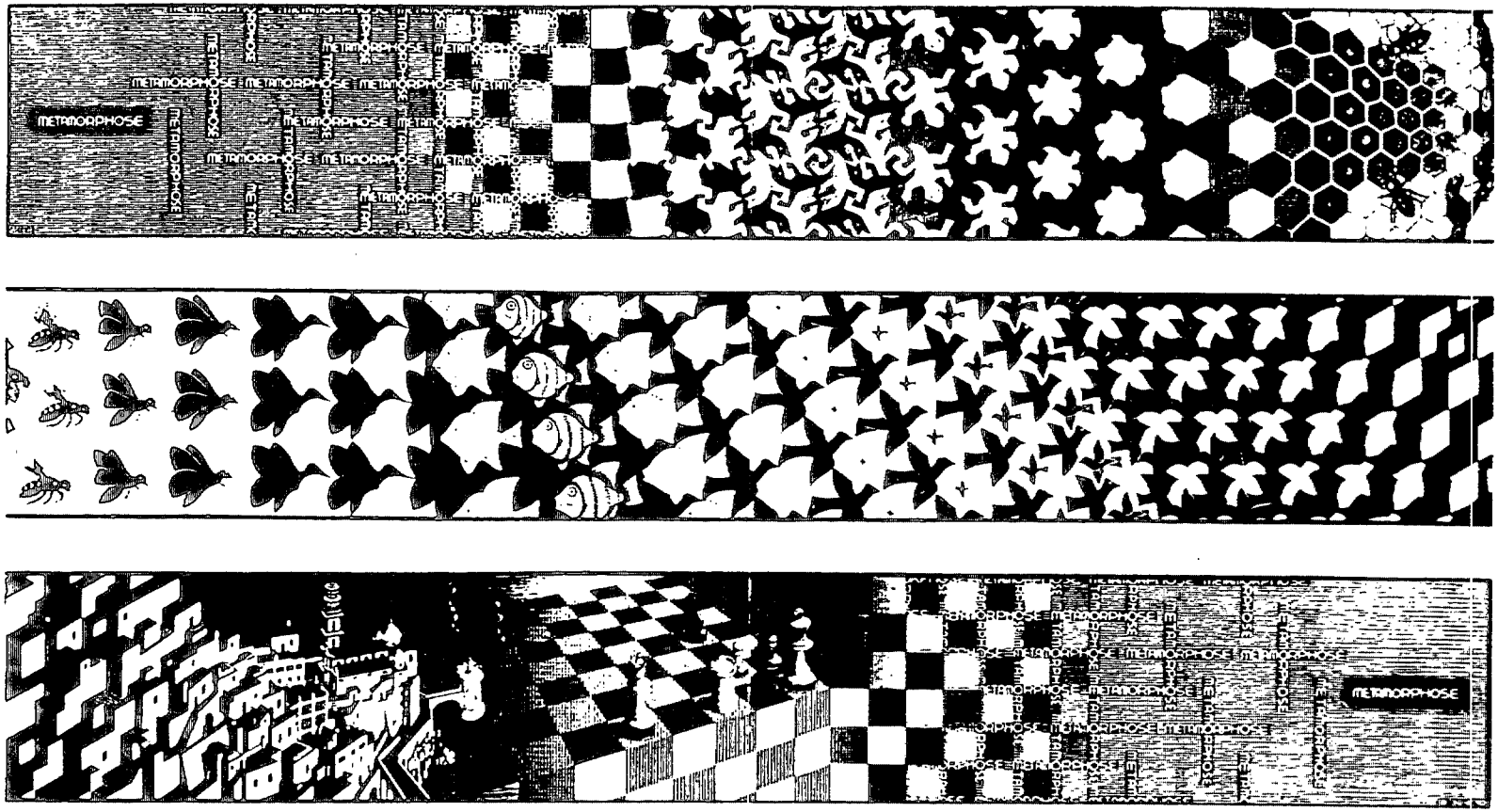
Vejledere: Peder Voetmann Christiansen
Karin Beyer

180/89 "KAOS I FYSISKE SYSTEMER eksemplificeret ved torsions- og dobbeltpendul".

af: Andy Wierød, Søren Brønd og Jimmy Staal

181/89 "A ZERO-PARAMETER CONSTITUTIVE RELATION FOR PURE SHEAR VISCOELASTICITY"

by: Jeppe Dyre




Metamorphosis II, *by M. C. Escher (woodcut, 19.5 cm. × 400 cm., 1939-40).*